IMAGE CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORKS ON MODIFIED MNIST DATASET

Aly Elgharabawy
260790925

Michael Li
260869379

William Zhang
260865382

Group 88

January 4, 2020

ABSTRACT

Recent development in convolutional neural networks (CNN) have made them the standard in computer vision tasks. In this assignment, we explore various CNN architectures and investigate their performance on a modified MNIST dataset. Image transformation techniques such as thresholding as well as data augmentation were applied to remove noise and regularize models. Transfer learning was also applied to implement state-of-the-art models trained on the ImageNet dataset. After hyperparameter tuning, the best model was an average ensemble of VGG16, Xception and Inception-ResNetV2, obtaining a final accuracy of 98.1%.

***K*eywords** Computer Vision · Machine Learning · Neural Network · Convolutional Neural Network · Modified MNIST · Transfer Learning

1 Introduction

Computer vision has historically been a task thought to be impossible to perform. However, there were eventually models that could perform acceptably but the breakthrough came from the work of Krizhevsky and his team on AlexNet [1], which demonstrated the effectiveness of deep learning models for computer vision, and started the "deep learning" era in machine learning. At the time, his model was considered huge with 5 convolutional layers, 3 max-pooling layers and 3 fully connected layers totalling over 60 million parameters. However, models with millions of parameters would quickly become the norm due to their superior performance.

For the assignment, our task was to classify a modified MNIST dataset, where each image was grayscale and would comprise of 3 numbers that are overlaid on a noisy background. The class of each image is based on the highest number shown within it. The task at hand was therefore to design or implement and validate a supervised classification model to perform the prediction task at the highest accuracy. We initially decided to design a convolutional neural network (CNN) from scratch, however, better results were found using pre-trained models. However, even better results were achieved when various pre-trained models were used together to classify each image. Each model was validated with a train-validation split of 80-20 and the final model was tested on the held out test set.

2 Related Works

As stated previously, AlexNet [1] truly influenced the way image classification was done, employing a large CNN and achieving a top-5 validation error rate of 16.4%, around 10% better than its competition in the 2012 ImageNet Large Scale Visual Recognition Challenge. Following such performance, many models were created using similar techniques of dropout for regularization and stacking convolutional layers to learn more complex features.

One of these models is the winner of the 2014 ImageNet challenge, called VGG Net and developed by Simonyan and Zisserman [2]. VGG Net achieved a top-5 error rate of 7.3% and employed 13 convolution layers, 5 pooling layers and 3 fully connected layers. This model had more than double the parameters compared to AlexNet, with 138 million parameters. The model demonstrated how small convolution filters (3x3) and deep networks (16-19 layers) were able to produce state-of-the-art results.

It was only a year later that He et al.[3] were able to achieve super-human performance on their model called ResNet. Their model obtained a top-5 error of 3.6%, which is 1.4% less than the human benchmark of 5%. Their model reformulated layers in order to reference layers that are not immediately precending it. This allowed for much deeper models, therefore greatly increasing their accuracy. This allowed ResNet to have 152 layers, about 8x more than VGG Net, while having less than half the parameters with slightly over 60 million.

3 Data Exploration

3.1 Original Data

The dataset comprised of 50,000 grayscale images with a resolution of 128x128. Figure 1 showcases 3 examples of images from the dataset.

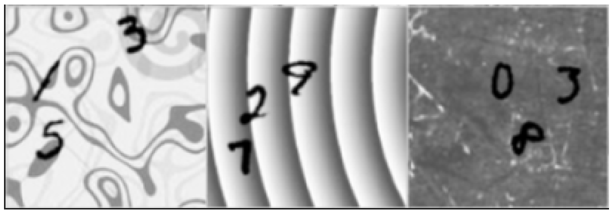


Figure 1: Examples of images in dataset

It is important to note that this 10-class image classification task is not balanced. This is natural since the task requires to identify the highest number in each image, it will naturally be skewed towards higher numbers. Figure 2 showcases the count of each class in the training set.

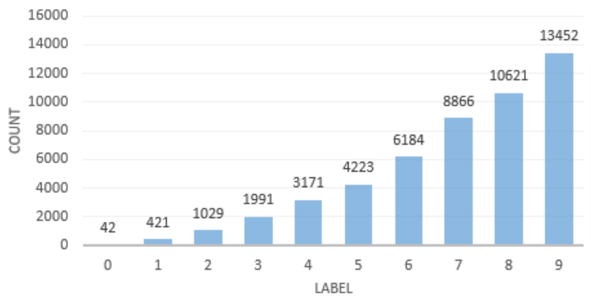


Figure 2: Distribution of label in training data

3.2 Preprocessing

Despite the fact that CNNs are able to filter out the noise in images, and should theoretically be able to ignore the noisy image, preprocessing the image to remove the background would aid the model in focusing solely on detecting all the numbers and finding the highest value.

Since the numbers were black pixels on a gray background, we employed a threshold technique to remove the background. To do so, we first normalized the pixel values, which was done by simply dividing by 255. Afterwards, we inverted the pixel values in order to have a black background and white numbers. Therefore, the background would have a value of 0 and be black while the digits would have a value of 255 and be white. Finally, we applied a threshold value of 0.925 which was determined empirically. The OpenCV python library was used to apply the threshold and remove any values that were above the threshold value, in this case anything that wasn't white. The middle column of figure 3 showcases two examples of preprocessed images.

3.3 Data Augmentation

Another issue with the data was that 50,000 training points would not be enough to train a deep CNN. For reference, ImageNet has over 15 million images. While each dataset was employed for a different task, it still demonstrates that 50,000 images is insufficient for training deep CNNs. Therefore, we employ a method called data augmentation, which generates new images that are slightly transformed from our original images. This technique helps avoid overfitting and helps the model generalize.

There are numerous transformations that can be applied to the images, such as rotation, translation and zoom. Since we are working with images containing numbers, the generated images did only had a small rotation amount. This is due to the fact that certain digits could have been transformed into another digit if the rotation was too wide. For example, a 6 could be rotated to a 9.

The package used to generate images is Keras' ImageDataGenerator, which generates new images every batch. These images are generated from the training set, therefore the original training images are never seen by the model.

Figure 3 showcases what an augmented image would look like, while some images may be cropped out due to the image being zoomed in, this helps the model since not all images may have whole numbers.

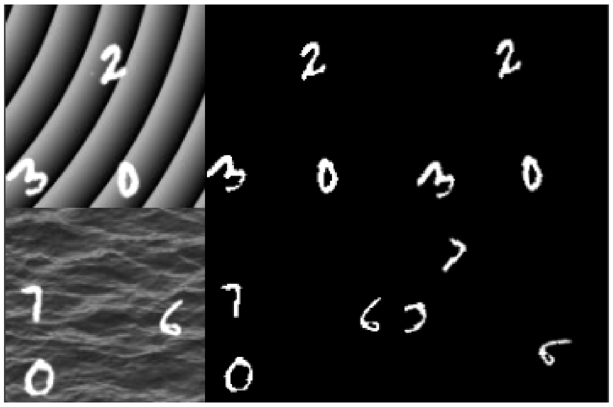


Figure 3: Comparison between original, processed and augmented images

4 Proposed Approach

A customized convolutional neural network was first tested on the dataset nananan.

Often, we need to solve a classification task in one domain of interest, but we only have sufficient training data in another domain of interest (i.e. ImageNet). However, the latter may be in a different feature space or following a different data distribution. Hence, the idea of transfer learning (a learning framework) is to adapt knowledge learnt in one setting to address a problem in another setting [4]. Three pre-trained models, namely VGG16, Xception and InceptionResNetV2, were tuned in order to adapt it to the modified mnist dataset. The latter were pre-trained on the ImageNet dataset which is a database that contains tens of millions of annotated images organized by the semantic hierarchy of WordNet [5]. The fully-connected layers of the pre-trained models were removed in order to train a new three fully-connected layers architecture that is adapted to the MNIST classification challenge.

4.1 Training and Fine-Tuning Pre-Trained Models

A specific strategy was employed to train and fine-tune the models. First, the convolutional base which contains convolutional and pooling layers is freezed to keep its original form. In fact, these pre-trained models contain rich and discriminative filters which can be adapted for our classification task. Then, the model is trained to initialize the fully-connected layers. This prevents the gradient from backpropagating random values through the network. Finally, all layers of the model are trained in order to fine tune the pre-trained model to the modified MNIST dataset.

4.2 Fully-Connected Layers Architecture

The following layers were added to the pre-trained models, a dense layer with 128 nodes, a dropout layer with a rate of 0.2, a dense layer with 64 nodes and a final layer with 10 nodes with a softmax activation function. All hidden layers use the relu activation function, as it has demonstrated high effectiveness proven by its wide use in almost all CNNs. Figure 4 shows the input and output dimensions of each layer.

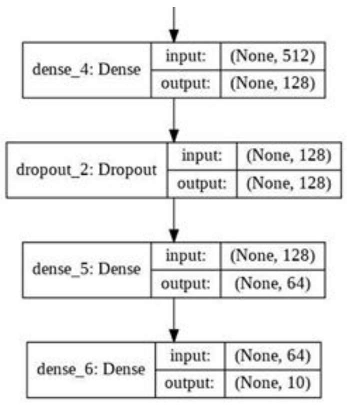


Figure 4: Last 4 layers of each pre-trained model adapted for the modified MNIST dataset

4.3 Validation

One fifth of the data was held out in order to validate our models. The other four fifths of the dataset are used to train the models using the data augmentation technique mentioned previously.

4.4 Regularization

One way our model applies regularization is by using data augmentation on the training set. The model never trains on the actual training set, and even though the augmented data is still derived from the training set, overfitting to the original data is still reduced. This reduces variance and therefore, improves our model's ability to generalize.

Another way we applied regularization to our model is by using the dropout layers in between the first and second stacked dense layers. [6] The dropout layer randomly drops hidden units at a specified probability. This prevents the model from fitting too much to one unit and allows it to focus more on other units, thereby reducing variance and overfitting.

4.5 Hyper Parameters

Often, a large batch size is preferred in order to speed up computations using the parallelism of GPUs. On the other side, Stochastic Gradient Descent could lead to overfitting when the batch size is equal to 1. Hence, the optimal learning rate, batch size and epochs were investigated for all models. At first, a batch size of 256 instances along with a learning rate of $1 * 10^{-3}$ was used to train the models on 30 epochs. However, we found that a bigger batch size leads to a lower asymptotic validation accuracy. Therefore, a batch size of 64 was used with the same learning rate and epochs was used to train the models. Finally, the learning rate was decreased to $5 * 10^{-4}$ in order to fine tune each model. In fact, the batch size is proportional to the learning rate. Hence, it is possible to obtain the same validation set accuracies under the same number of epochs. This reduces the number of parameter updates and shorter training times [8].

4.6 Loss Function

All models were trained on categorical cross entropy. This is the standard loss function for multi-class and single-label classification. The loss is calculated using equation 1,

$$J(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i) \tag{1}$$

where N is the number of classes, y_i is the actual label and $\hat{y}_i$ is the predicted label. This is loss is divided by the number of classes, to make it invariant to the amount of classes.

However, since our labels were integers, we used the sparse categorical cross entropy, which follows the same formula, but is more efficient in memory usage. For example, in standard categorical cross entropy, the labels would be one-hot encoded, and stored as shown in equation 2. However, since we are using sparse categorical cross entropy, it will be stored as shown in equation 3.

$$\begin{aligned} \mathrm{y}_{pred} = [&[0,0,1],\\ &[0,1,0],\\ &[0,1,0]] \end{aligned} \tag{2}$$

$$\mathrm{y}_{pred} = [2,1,1] \tag{3}$$

4.7 Optimizer

Another key choice for our training process was the optimizer, which dictates the changes to be applied for gradient descent. Since the optimizer is in control of optimizing many parameters in a non-convex problem, the optimizer will be liable to issues such as gradient vanishing as well as oscillation.

For the optimizer, we chose Stochastic Gradient Descent [9] to facilitate training on small batch sizes, and therefore enable us to optimize better through smaller gradient steps. In addition, we chose to add momentum for our optimizer, which significantly counteracts gradient vanishing at local minima. Instead of gradient steps being calculated using only the current gradient, the previous update is taken into account and is multiplied by its separate learning rate as shown in equation 4.

$$\Delta w = \alpha \Delta w - \eta \nabla Q_i(w) \tag{4}$$

where Δ w is the update to be applied to the weights, α is the learning rate, η is the momentum coefficient and $\nabla Q_i w$

Factoring the previous update into the next update allows the optimizer to get past local minima and mitigate oscillations, ensuring that it has a predisposition to update in the direction the previous update took. This ultimately gives it a push-forward effect.

5 Model Selection and Results

5 different models were tested, their validation accuracy, run time per epoch and their validation loss were compiled into table 1. All training and validation accuracy and losses were compiled into figures in the appendix.

Model	Accuracy	Runtime/Epoch	Loss
VGG	97.67%	152s	0.112
Xception	97.12%	188s	0.180
Inception-ResNetV2	97.8%	331s	0.199
Average	98.9%	-	-
Stacking	98.4%	171s	0.031

Table 1: Comparison of model accuracy and runtime

5.1 VGG16

Developed by Simonyan and Zisserman [2], VGG16 is considered as a great initial model to start with due to its homogeneous design. The model is composed of 13 convolution layers and 5 pooling layers. A very interesting approach of VGG is stacking convolutional layers with small filter sizes (3x3) in order to extract complex features, while keeping computation load low.

After 30 epochs of training, VGG16 was able to achieve 97.67% validation accuracy, and a test accuracy of 97.47% on the kaggle competition.

5.2 Xception

Xception, developed by Chollet [10], expands on the traditional architecture of convolutional neural networks. Xception is based on the Inception model created by Szegedy et al [11]. While VGG demonstrated how increasing the depth of convolutional neural networks increased their ability to extract complex features, Inception showcased the benefits of increasing the width of a model. See appendix figure 13 for an breakdown on the convolution layer expansion.

Xception, known as Extreme Inception, further expands the concept by employing depth wise separable convolutions. The underlying assumption in Xception is that cross-channel correlations and spatial correlations are independent. The model uses a set of 1x1 convolutions to map the data into separate spaces and having various convolutions applied to those spaces. Appendix figure 14 illutrates this principle.

Using a version of Xception trained on ImageNet and chaging the last dense layers, the model obtained a validation accuracy of 97.12%, which is marginally worse than VGG16, despite taking more epochs to train using the same hyperparameters.The model achieved a test accuracy of 97.33% on the kaggle competition.

5.3 Inception-ResNetV2

Inception-ResNet, developed by Szegedy [11], included the concept of residual networks with the inception model. Residual networks were first explored by He et al. [3], in which they demonstate that increasing the depth of neural networks have diminishing returns due to the difficulty of learning direct mappings. To circumvent this issue, residual networks learn the difference, or the residual mapping. Appendix figure 12 illustrates this concept.

Our model trained on top of Inception-ResNetV2. At the beginning, it was very slow due to the big size and high number of parameters. However, after extended training while manually decaying the learning rate, it managed to plateau at a validation accuracy of 97.8%, and a test accuracy of 97.5% on the kaggle competition.

5.4 Average Ensemble

Even though these models produced very strong results in the given task, each model still had its shortcomings, and there was room for improvement. One way to combat these shortcomings is to combine these models' results by averaging. This technique makes the models complement each others' weaknesses by adding each model's prediction for each class and averaging the result. This way, if a model has a weakness or uncertainty in a certain area that the other two do not, that shortcoming will be covered. This averaging operation therefore reduces variance by reducing the impact one false prediction would have on the final prediction, due to the division operation.

The ensemble managed to achieve 98.9% validation accuracy, and a 98.1% test accuracy on the kaggle competition.

5.5 Stacking Ensemble

Another way to ensemble the models together is to use stacking: A technique that uses a fully connected layer that gets fed the models' predictions and outputs a new set of predictions. The fully connected layer learns the weights it would assign to each set of predictions, calculates an error based on its results, and then performs gradient descent to adjust its weights accordingly. Following that procedure, the model learns which combination of predictions it could use to minimize error.

The ensemble managed to achieve 98.4% validation accuracy, but achieved a 97.8% test accuracy on the kaggle competition.

5.6 Test Results

Model	Test Accuracy
VGG	97.43%
Xception	97.33%
Inception-ResNetV2	97.5%
Average	98.10%
Stacking	97.76%

Table 2: Comparison of model test set accuracy

Of all the models scores on the kaggle test set, the average ensemble model performed the best with a test accuracy of 98.10%, followed by the stacking ensemble with an accuracy of 97.76%. This can be attributed to the models compensating for one another's errors and therefore leading to variance reduction and accuracy improvement. The ensemble model also learns a better inductive bias from the combination of its constituents' biases.

6 Conclusion and Discussion

In this task, we managed to utilize 3 different convolutional neural networks originally made for a different classification task and adapted them to fit our task, using 3 dense layers stacked on top of the original model. The models used have proven to be highly effective and versatile. The extent and limitations of this versatility should definitely be investigated further, since transfer learning has proven to be a powerful tool in this experiment, especially when combined with ensemble techniques.

While all these models were trained on another image classification task, it would be interesting to see the performance obtained if the initial training was done on the modified MNIST dataset.

7 Statement of Contribution

- Aly Elgharabawy: Model research and training, hyperparameters optimization.
- Michael Li: Data pre-processing, model research and training
- William Zhang: Documentation, model statistics

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, in Advances in neural information processing systems, 2012, pp. 1097-1105.

[2] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*. arXiv preprint arXiv:1409.1556, 2014.

[3] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition* in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770-778.

[4] S. J. Pan and Q. Yang, *A survey on transfer learning*. IEEE Transactions on knowledge and data engineering, vol. 22, no. 10, pp. 1345-1359, 2009.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, *Imagenet: A large-scale hierarchical image database*. 2009 IEEE conference on computer vision and pattern recognition, 2009: Ieee, pp. 248-255.

[6] *Srivastava, N., et al. (2014). "Dropout: a simple way to prevent neural networks from overfitting J J. Mach. Learn. Res." 15(1): 1929-1958.*. Proceedings of the 30th International Conference on Machine Learning.

[7] Sutskever, I., et al. (2013). *On the importance of initialization and momentum in deep learning*. Proceedings of the 30th International Conference on Machine Learning. l. (2018).

[8] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le, *Don't decay the learning rate, increase the batch size*. arXiv preprint arXiv:1711.00489, 2017.

[9] Kiefer, J. and J. Wolfowitz (1952). *Stochastic Estimation of the Maximum of a Regression Function*. The Annals of Mathematical Statistics 23(3): 462-466.

[10] Chollet, F. (2017), *Xception: Deep Learning with Depthwise Separable Convolutions*. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[11] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, *Inception-v4, inception-resnet and the impact of residual connections on learning*. Thirty-First AAAI Conference on Artificial Intelligence, 2017.

Appendix

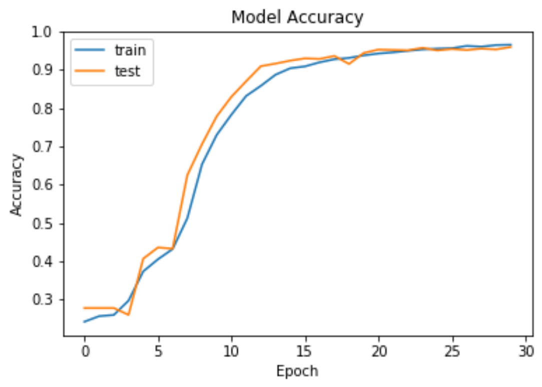
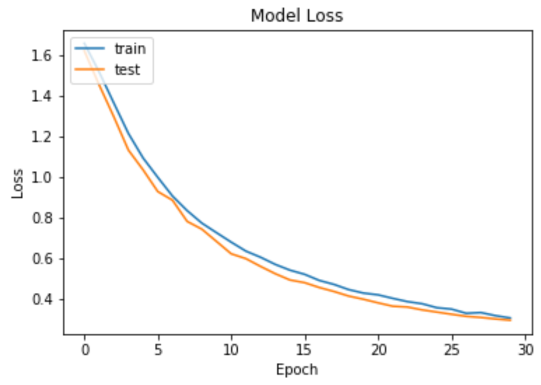


Figure 5: VGG accuracy over 30 epochs



1

Figure 8: Xception loss over 30 epochs

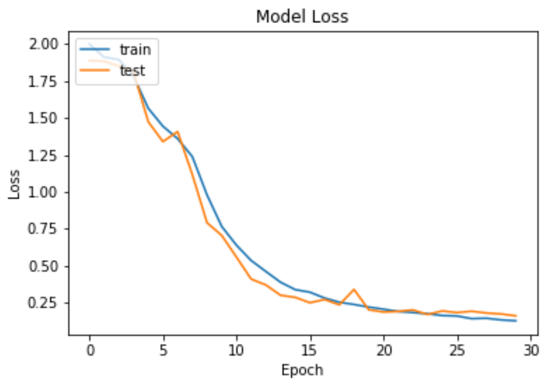


Figure 6: VGG loss over 30 epochs

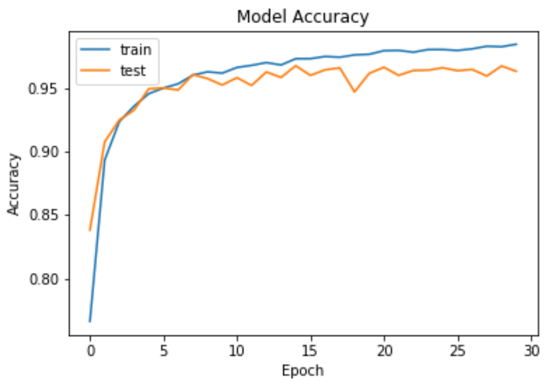


Figure 9: InceptionResNetV2 accuracy over 30 epochs

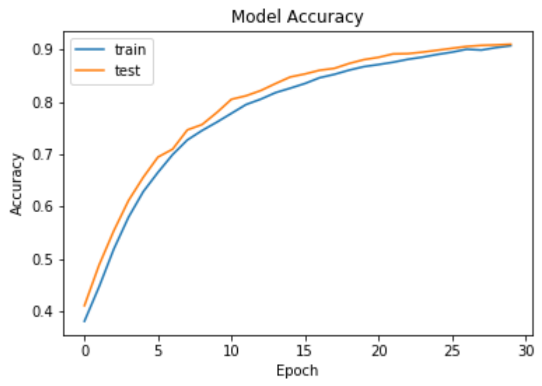


Figure 7: Xception accuracy over 30 epochs

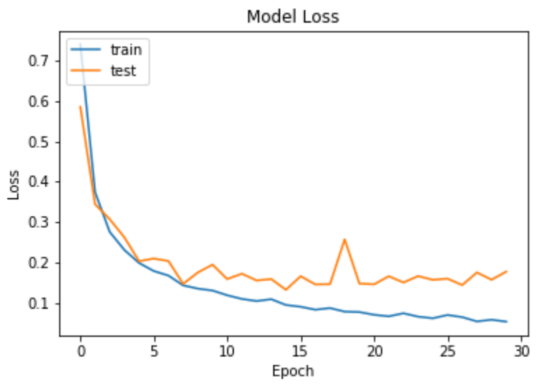


Figure 10: InceptionResNetV2 loss over 30 epochs

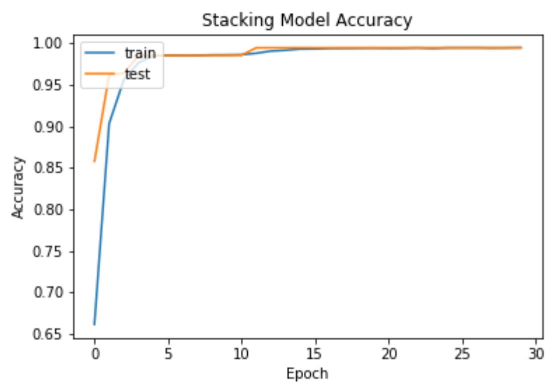


Figure 11: Xceptions accuracy over 30 epochs

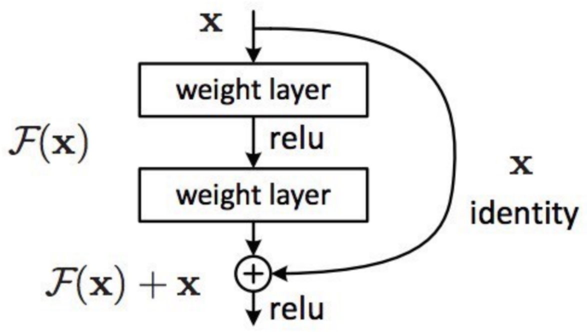


Figure 12: Residual Block

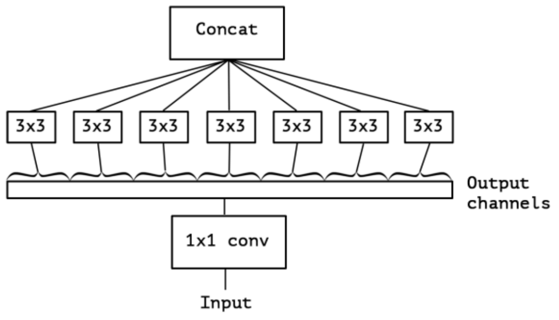
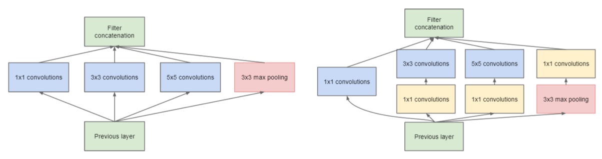


Figure 13: Module used in Xception model



(a) Inception module, naïve version (b) Inception module with dimension reductions

Figure 14: Module used in inception network