Computers, Materials & Continua
DOI:10.32604/cmc.2021.015318
Article

Tech Science Press

An Optimized Deep Residual Network with a Depth Concatenated Block for Handwritten Characters Classification

Gibrael Abosamra* **and Hadi Oqaibi**

Department of Information Systems, King Abdulaziz University, Jeddah, 21589, SA
*Corresponding Author: Gibrael Abosamra. Email: gabosamra@kau.edu.sa
Received: 15 November 2020; Accepted: 24 December 2020

Abstract: Even though much advancements have been achieved with regards to the recognition of handwritten characters, researchers still face difficulties with the handwritten character recognition problem, especially with the advent of new datasets like the Extended Modified National Institute of Standards and Technology dataset (EMNIST). The EMNIST dataset represents a challenge for both machine-learning and deep-learning techniques due to inter-class similarity and intra-class variability. Inter-class similarity exists because of the similarity between the shapes of certain characters in the dataset. The presence of intra-class variability is mainly due to different shapes written by different writers for the same character. In this research, we have optimized a deep residual network to achieve higher accuracy *vs.* the published state-of-the-art results. This approach is mainly based on the prebuilt deep residual network model ResNet18, whose architecture has been enhanced by using the optimal number of residual blocks and the optimal size of the receptive field of the first convolutional filter, the replacement of the first max-pooling filter by an average pooling filter, and the addition of a drop-out layer before the fully connected layer. A distinctive modification has been introduced by replacing the final addition layer with a depth concatenation layer, which resulted in a novel deep architecture having higher accuracy *vs.* the pure residual architecture. Moreover, the dataset images' sizes have been adjusted to optimize their visibility in the network. Finally, by tuning the training hyperparameters and using rotation and shear augmentations, the proposed model outperformed the state-of-the-art models by achieving average accuracies of 95.91% and 90.90% for the Letters and Balanced dataset sections, respectively. Furthermore, the average accuracies were improved to 95.98% and 91.06% for the Letters and Balanced sections, respectively, by using a group of 5 instances of the trained models and averaging the output class probabilities.

Keywords: Handwritten character classification; deep convolutional neural networks; residual networks; GoogLeNet; ResNet18; DenseNet; drop-out; L2 regularization factor; learning rate

1 Introduction

Handwritten character recognition has a wide spectrum of applications in all software systems that allow handwritten input through an electronic stylus or digital tablet or through an offline scan of documents such as postal envelopes, bank cheques, medical reports, or industrial labor reports. The handwritten alphanumeric character recognition problem is still a challenge in the field of pattern recognition due to the high variability within each handwritten class and the high similarity among certain classes in the alphabet itself. The EMNIST dataset [1] is considered one of the recent datasets that possess such difficulties where there is variability within each class due to the variability of the patterns produced by different writers and variability for the same writer who produces different patterns for the same character due to the nature of the handwriting process itself. Traditional machine-learning techniques have been developed to tackle this challenge, leading to accuracies that may be accepted for old datasets such as the Modified National Institute of Standards and Technology dataset (MNIST). Recently deep-learning techniques, especially Deep Convolutional Neural Networks (DCNN), made a breakthrough that upgrades the recognition accuracy of machines to an accuracy level comparable to a human being's recognition accuracy. The MNIST dataset is now considered an already solved problem where the percentage error of many works approaches 0.2% that is considered to be irreducible. The EMNIST dataset is built to represent the new challenging handwriting alphanumeric dataset instead of the old MNIST dataset. The EMNIST is an extended version of the MNIST dataset [2], which consists of 52 characters (both upper and lowercase) and 10 digits. It includes a total of 814255 samples from almost 3700 writers [1]. The dataset has different labeling schemas: (1) By_class (62 alphanumeric characters: digits: 0–9, upper case: A–Z, and lower case: a–z), (2) By_merge (47 classes: digits 0–9, 37 fused upper and lower case letters where similar upper and lower letters are unified), (3) Letters (26 classes having one class for each letter, either upper or lower), and (4) Digits (10 classes: 0–9 with more and different samples than MNIST digits). A Balanced dataset section is introduced that represents a reduced version of the By_merge section but has an equal number of samples for each label. A limited number of systems have been built to reach significant classification accuracies of the different sections of the EMNIST dataset, as will be shown in Section 2.

Our approach is mainly based on starting with the architectures of two main ready-made image classification DCNN models and modify them by exploring the possible dimensions of variations such as increasing or decreasing the number of building blocks (or layers) to select the faster and more accurate one, and then enhance the selected architecture. Also, the suitable pre-processing and augmentation of the image data is accompanied to get the highest possible accuracy to compete with state-of-the-art results.

In Section 2, we summarize and record the state-of-the-art results of the most important researches done on the Letters and Balanced sections of the EMNIST dataset. In Section 3, a brief background of DCNN and the smallest DCNN models that we think have significant achievement in the image classification problem is presented. In Section 4, our work is elaborated with detailed experiments, in addition to the presentation of the different types of image preprocessing and data augmentation that are considered, until reaching a final configuration of a DCNN that gives competing accuracy with state-of-the-art systems. In Section 5, final experiments are performed on the Letters and Balanced sections of the dataset, and two extra experiments are performed on each section using a reduced training set for each dataset section. In Section 6, our work is compared with state-of-the-art works. Finally, important contributions are summarized in the conclusions and future work section.

2 Literature Review

In this section, we concentrate on CNN, Capsule network (CapsNet), and machine-learning based efforts done after the creation of the EMNIST dataset [1]. Works done on the National Institute of Standards and Technology dataset (NIST) Special Database 19 (from which EMNIST was obtained) can be found in [3]. Also, we mainly refer to works done on the Letters and Balanced sections only.

The authors of the original EMNIST paper [1] included a baseline using a linear classifier and the Online Pseudo-Inverse Update Method (OPIUM) [4]. Using OPIUM, the highest baseline reported accuracies were 85.15% for the Letters dataset and 78.02% for the Balanced dataset.

The authors of [5] combined Markov random field-models with CNN (MRF-CNN) for image classification. They reported accuracies of 95.44% and 90.29% when applying their technique to the Letters and Balanced datasets, respectively.

The authors of [6] used K-Nearest Neighbor (KNN) and Support Vector Machine (SVM) classifiers to classify handwritten characters based on features extracted by a hybrid of Discrete Wavelet Transform (DWT) and Discrete Cosine Transform (DCT). They reported accuracy of 89.51% when their model was applied to the Letters data set (using SVM based on DWT's and DCT's combined features).

In [7], a bidirectional neural network (BDNN) was designed to perform both image recognition and reconstruction using an added style memory to the output layer of the network. When their model was tested on the EMNIST Letters dataset, it achieved an accuracy of 91.27%.

Some authors have made use of neural architecture searches to automatically optimize the hyper parameters of CNN classifiers. For example, Dufourq et al. [8] introduced a technique called Evolutionary Deep Networks for Efficient Machine Learning (EDEN). They applied neuro evolution to develop CNN networks for different classification tasks achieving an accuracy of 88.3% with the EMNIST Balanced dataset. Researchers in [9] used genetic algorithms in the automatic design of DCNN architectures (Genetic DCNN) for new image classification problems. When their generated architecture was applied to the Letters dataset, it achieved an accuracy of 95.58%, which is the highest published accuracy to date.

Committees of neuro evolved CNNs using topology transfer learning have been introduced by the authors [10], who obtained an accuracy of 95.35% with the EMNIST Letters dataset. Due to the highly challenging level of this classification task, they resorted to the use of 20 models to enhance accuracy from 95.19% to 95.35% which was still less than the highest previously recorded accuracy (95.44%).

Researchers [11] introduced a hierarchical classifier that uses automatic verification based on a confusion matrix extracted by a regular (flat) classifier to enhance the accuracy of a specific classifier type. It relatively succeeded in enhancing the accuracy of some machine-learning based techniques such as the linear regression classifier but not in enhancing the accuracy of the CNN classifier. Its flat CNN classifier resulted in an accuracy of 93.63% and 90.18% when applied to the Letters and Balanced datasets, respectively.

The researchers [12] proposed TextCaps, which used two capsule layers (a highly advanced concept introduced by Sabour et al. [13]) preceded by 3 CNN layers. They reported accuracies of 95.36% and 90.46% for EMNIST Letters and EMNIST Balanced datasets, respectively.

Some researchers designed DCNN architectures to classify letter sets other than the English alphanumeric handwritten characters and then tested their models in the EMNIST database.

For example, the researchers [14] built a DCNN for the classification of Arabic letters, and when they tested their architecture on the EMNIST Letters dataset, they reported accuracy of 95%. Similarly, the authors of [15] designed a system using a DCNN with six CNN layers and two fully connected layers for Bangla handwritten digit recognition. They reported an accuracy of 90.59% when their DCNN model was tested on the EMNIST Balanced dataset. Fig. 1 displays a timeline of the accuracy recorded for the Letters dataset in the mentioned references based on their published dates.

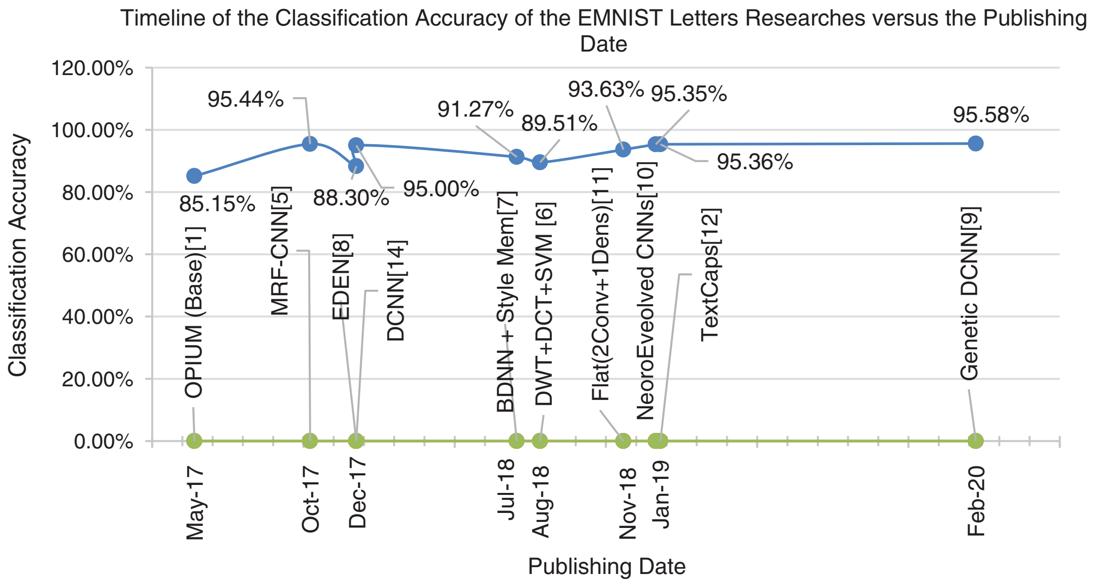

Figure 1: A timeline graph of the accuracies achieved by different researches on the Letters dataset

From Fig. 1, it is clear that DCNN-based techniques resulted in higher accuracies compared with machine-learning techniques or non-deep learning techniques when applied to the EMNIST Letters dataset. Also, some techniques more advanced than DCNN did not achieve remarkable accuracies such as the CapsNet (TextCaps [12]) although characterized by orientation-invariability and requires less number of training samples.

From the timeline curve, we also observe two global peaks. The first significant peak was achieved in October 2017 by the MRF-CNN technique [5] with a 10.29% increase from the base classifier [1]. All subsequent works from October 2017 until February 2020 didn't even reach the achieved accuracy of 95.44% (in October 2017). The second peak (95.58%) was achieved after approximately two-and-a-half years by the Genetic DCNN technique [9], which raised accuracy by 0.14%.

The timeline includes two very close local peaks (there is mathematically one peak, but because their accuracies are very close we can consider them two in one) representing two highly advanced techniques ([10,12] published on 1 January 2019 and 9 January 2019, respectively), which have a 0.01% difference in accuracy.

The same conclusions are drawn when investigating the achievements in the case of the Balanced dataset, but with lower accuracy due to the inclusion of more classes (47) than the Letters dataset (26 classes), which resulted in more confusing letter shapes.

An important conclusion to be drawn from this review is that no advanced architecture of the DCNN [16] (such as ResNet, Inception, DenseNet) has been applied to the EMNIST Letters or Balanced datasets in standalone research. The researchers [9] tested some ready-made architectures without modifications on the EMNIST letters yielding accuracies of 89.36%, 94.62%, and 94.44% using AlexNet, VGGNet, and ResNet, respectively. Hence, in this paper, we will introduce standalone research to optimize one of these advanced architectures, specifically the ResNet architecture, to achieve higher classification scores on the Letters and Balanced datasets.

3 Review of the Concerned Convolutional Networks

In this section, we summarize the basic building blocks of DCNN and the two most famous DCNN networks, mainly GoogLeNet and ResNet18 and the main concepts on which these networks are based, in addition to a brief definition of the drop-out technique.

3.1 Deep Convolutional Neural Networks (DCNNs)

A typical DCNN is composed of many convolutional layers, max-pooling layer and/or average-pooling layers, one or two fully connected layers and finally a softmax layer in the case of classification tasks. The purpose of a convolutional layer is to map an M-channel Input (image) of size $I_w \times I_h$ into N-channel output (Feature Maps (FMs)) with size $O_w \times O_h$ by using N Convolutional Filters (CFs) of size $F_w \times F_h$. Each of the N CFs is applied to each of the overlapping windows (having size $F_w \times F_h$) in the M-channel input layer using dot product operations to generate one of the N-channel outputs. The overlapping between the windows is controlled by the step length (stride) in the horizontal and the vertical directions when the filter is iterated across the windows of the input channels that affects the size of the output FMs ($O_w \times O_h$). The max-pooling layer (or average-pooling layer) is a special filter with size ($P_w \times P_h$) that is applied to each of the overlapping windows (of size $P_w \times P_h$) in every individual input channel to map it into its corresponding output channel using max (or average) operation with a stride length of 2 or more to perform a type of down-sampling to reduce the size of the output channels (FMs). Down-sampling can also be achieved by the CFs when using a stride of 2 or more. The convolutional layers are always followed by activation layers that map each output value (v) to values that fall within the ranges of (-1 to $+1$), (0 to 1), or (0 to v when $v > 0$ and 0 otherwise). The fully connected layer maps the last KFMs into C outputs in case of C classes or C regression outputs. The softmax layer is used for classification purposes to determine the output classes' relative probabilities based on the input real values such that all the output probabilities sum to 1. All the learnable weights of the DCNN are initialized randomly using a variety of techniques. By training a DCNN network using the proper backpropagation algorithm, the weights of the CFs and the fully connected layers are optimized for a specific classification or regression task. When many convolutional layers are cascaded, the first CFs do a role similar to image processing filters such as smoothing and edge detection; however, in deeper levels, the CFs perform different types of abstractions until the required output representation is reached. Fig. 2 depicts a simple DCNN.

3.2 GoogLeNet

As described in the previous subsection, in a convolutional operation at one location, every output channel (N) is connected to every input channel (M), so it is called a dense connection architecture. The GoogLeNet [17] builds on the idea that most of the activations in a deep network are either unnecessary (value of zero) or redundant because of correlations between them. Therefore, the most efficient architecture of a deep network will have a sparse connection between the input and output activations, which implies that all N output channels (FMs) will not have a connection with all the M input channels (FMs). There are techniques to prune out such connections which would result in a sparse weight/connection. However, kernels for sparse matrix multiplication are not optimized in libraries such as BLAS or CuBlas (CUDA for GPU) packages, which render them even slower than their dense counterparts. So, GoogLeNet devised an inception module that approximates a sparse CNN with a normal dense construction (as shown in Fig. 3).

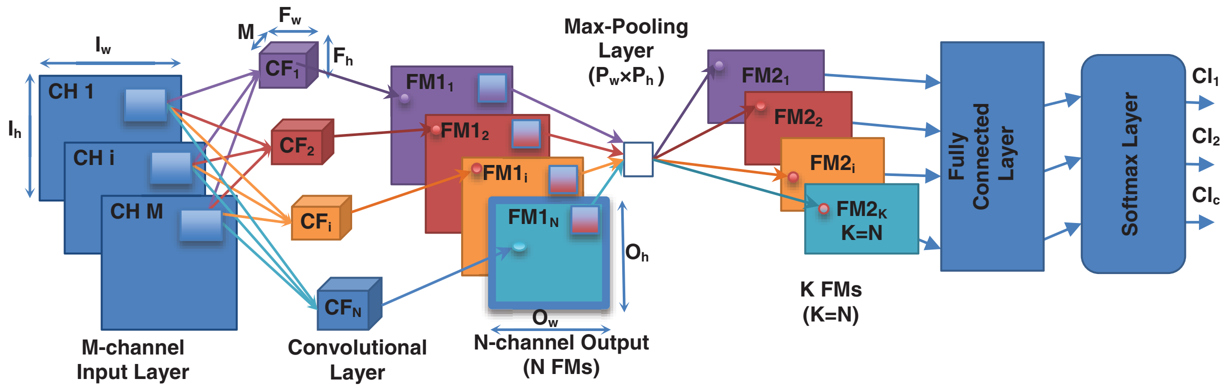

Figure 2: A simple DCNN composed of an input layer, a convolutional layer, a max-pooling layer, a fully connected layer, and a softmax layer

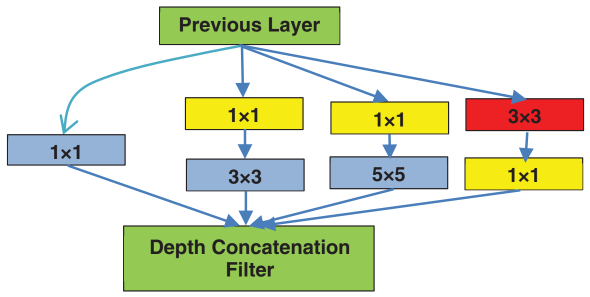

Figure 3: Inception module with dimension reduction using 1×1 convolutions [17]

Since only a small number of neurons are effective as mentioned earlier, the width/number of the CFs of a particular kernel size is kept small. Also, it uses convolutions of different sizes to capture details at different scales (5×5, 3×3, 1×1).

Another salient point about the inception module is that it has a so-called bottleneck layer (1×1 convolutions as shown in Fig. 3). It helps in the massive reduction of the computation requirements.

The GoogLeNet model is built of 9 inception modules in addition to the lower layers of traditional convolution and max pooling and the final layers for average pooling, drop-out, linear mapping (fully connected), and soft-max layer for classification. All the convolutions, including those inside the inception modules, use rectified linear activation. By running multiple instances of the GoogleNet in conjunction with several types of data augmentation and by aggregation of their class probability outputs, the GoogleNet team achieved a top-5 error of 6.67% in the ILSVRC 2014 competition.

3.3 ResNet18

Residual networks are deep networks that solve the degradation problem by letting a layer or group of layers learn the residual of a mapping function instead of learning the complete mapping function by making the output mapping equal to the summation of the input plus the learned function [18]. In other words, if we assume that the input is a simple variable X and the output is the required mapping MP(X), the learned function F(X) will be

$$F(X) = MP(X) - X \tag{1}$$

This is accomplished simply by using a direct shortcut connection from the input and adding it to the output as shown in Fig. 4, which includes two mapping layers, each followed by a Rectified Linear Unit (ReLU).

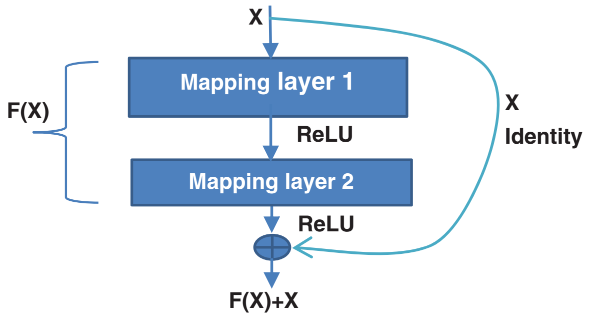

Figure 4: A basic residual learning building module (or block) [18]

When N residual modules are cascaded, the final residual output ResOut(N) without regarding the intermediate non-linear effects can be defined iteratively by the following formula:

$$\mathrm{ResOut(N)} = \mathrm{Input}_0 + \sum_{i=1}^{N} \mathrm{ResOut(i)} \tag{2}$$

where Input_0 is the input to the first residual module.

Which can be represented by the following recursive formula:

$$\mathrm{ResOut(N)} = \begin{cases} F_N(\mathrm{ResOut(N-1)}) + \mathrm{ResOut(N-1)} & N > 0 \\ \mathrm{Input}_0 & N = 0 \end{cases} \tag{3}$$

where F_N is the learned function at block N.

A very important characteristic of residual networks that makes them compete with inception networks is that a residual network that includes many residual modules with intermediate down sampling between every two modules has the advantage of combining different scales of the input image using addition in an iterative manner rather than combining them using concatenation in a parallel manner in the inception networks. Thus, the residual architecture allowed deeper networks (such as ResNet18) to learn better than plain networks and hence achieved higher accuracies on many well-known datasets such as ImageNet and CIFAR-10 than previous plain networks [18]. The ResNet18 model is built of 8 residual modules preceded by lower layers of traditional convolution and max-pooling and followed with final layers for average pooling, linear mapping (fully connected), and a soft-max layer for classification. Batch normalization [19] is adapted right after each convolution and before rectified linear activation.

3.4 The Drop-Out Technique

Overfitting is a serious problem in a deep neural network, especially when a large set of parameters is used to fit a small set of training data. Drop-out is a powerful technique to address this issue [20]. This works by randomly removing neurons at a fixed probability during training, and then using a whole architecture at test time. This counts as combining different "thinned" subnets for improving the performance of the overall architecture. Drop-out is used in GoogLeNet, while it is not used in ResNet18. We will add a drop-out layer in the residual DCNN solution to test its effect on classification accuracy.

4 Development of the Proposed Solution Through Experimentation

In this section, we start implementing our methodology by experimenting and comparing the two lightweight prebuilt models GoogLeNet and ResNet18 in the classification of the EMNIST Letters dataset with demonstration of important types of image preprocessing and data augmentation methods that can help in increasing classification accuracy.

Then, the architectures of both models (ResNet18 and GoogLeNet) are modified by the removal or the addition of a limited number of their basic building blocks, and the effects of the modifications are tested based on the resultant accuracy.

Finally, fine-tuning of the selected architecture from the previous step is achieved through some architecture modifications such as the insertion of a drop-out layer (with proper drop-out probability) and the replacement of one or more addition layers with depth concatenation layer(s), in addition to the selection of the best augmentation methods and the optimum values for the training hyper parameters such as learning rate and regularization factor. It is important to note that due to the stochasticity involved in the training process, every experiment is repeated three times, and the mean accuracy of the three generated models is used to identify the best architecture modification or parameter value that will be implemented in the final solution(s). Only the final solutions are repeated fifteen times, and their mean accuracies are calculated and recorded for comparison with state-of-the-art results. Since we have based our methodology on using a pre-built DCNN that has already been optimized for recognizing images and our goal is to adapt this architecture to recognize handwritten character images, we use a concept similar to partial differentiation by changing a single parameter and observing its effect and using the best one or two values for this parameter and continue in this manner using a beam search with a beam width of two in most cases until reaching to optimal or semi-optimal tuning of the used architecture. The reason for using a beamwidth of two is that the tested parameters may have a correlation and also due to the stochasticity of the training process. Increasing the beam width to 3 or 4 may be necessary in some cases to guarantee that we didn't miss a significant solution path, especially when there is a structural modification such as adding (or removing) layers (or blocks) or changing the way of merging two layers from addition to concatenation. Also, using a beamwidth of 1 is possible if the best path is very clear.

4.1 Preprocessing and Data Augmentation

Although, the dataset images are well prepared, some preprocessing may be needed to enhance the visibility of the images to the deep network model used in this research. Also, the adaption of the input images to the required input characteristics of these models such as resizing and gray-to-color conversion is performed whenever necessary.

4.1.1 Increasing the Background Area Around the Bodies of the Characters

Although the bodies of the characters are centered in the images of the characters of the EMNIST dataset, some character bodies may extend to touch the end of the boundary of their images. Increasing the background area (surrounding blank pixels) around the bodies of the characters allows the bodies of the characters (or their transformed versions) to be processed by the CFs without being affected by the padding mechanism needed at the boundary of the images, either in the earlier layers of the network or the deeper layers. In addition, during image augmentations such as rotation, the resultant images may suffer from distortion if the rotated size of the character body exceeds the original image size.

Increasing the background area around the bodies of the characters minimizes such effects. For example, if the character body is represented by zeros and the background by ones and the character stroke touches the extreme ends of the rectangle surrounding the character, then zero padding will increase the thickness of the character stroke by the amount of padding.

By adding a blank padding of 6 pixels per side (for example), the character image size becomes 40×40 pixels instead of 28×28 pixels, which makes the boundaries of the bodies of the characters less affected by the filters padding mechanisms. In addition to avoiding the padding effect, the extra blank padding controls the percentage of the character body in the resized image seen by the DCNN model. Hence, allowing accurate adaptation by selecting the appropriate zooming factor to the available mapping and processing done by the concerned model, as will be verified by the performed experiments.

4.1.2 Image Resizing and Gray-to-Color Conversion

Image resizing is done using bi-cubic interpolation [21]. Grey-to-color conversion is done simply by using three copies of each grey image to represent the three channels of the color version to adapt the image with the input characteristics of the used network model. This type of conversion is needed only when training the original model with its pre-trained parameters on the new images dataset in a transfer-learning paradigm.

4.1.3 Image Augmentation

Although, the number of training samples in each EMNIST dataset section is large, some type of data augmentation is needed to increase the trained model generalization when subjected to new unseen samples in the testing phase.

Since we deal with handwritten characters, image rotation and shearing [22] are beneficial augmentation methods because they increase the possible views of the input data and hence increase the network ability for generalization, as will be shown in the early and fine-tuning experiments.

4.2 Testing ResNet18 and GoogLeNet

In this subsection, GoogLeNet and ResNet18 prebuilt DCNN models are tested through transfer learning on the EMNIST Letters dataset. The models are used in two cases with and without their pre-trained weights.

Since these prebuilt networks were built to classify color images with different sizes, the Letters dataset images are resized, and each image is copied 3 times to represent the 3-channel images to be compatible with the required input of the concerned DCNN model. In the case of using the models without their pre-trained weights, the inputs of the models are modified to accept 1-channel images, as will be done in the following subsections.

4.2.1 Testing ResNet18 Model

This model accepts 3-channel images with a size of 224×224 pixels. Hence, each image is resized to this size and copied 3 times to represent the three channels. Since the original network was built to classify 1000 classes, only the last two layers are replaced by new ones, mainly the fully connected layer and the softmax layer where only 26 classes are present in the EMNIST Letters dataset.

Training of the modified net is done using Stochastic Gradient Descent with Momentum (SGDM) optimizer [23] with a base learning rate of 0.1 and learning rate drop factor of 0.1 applied every 6 epochs for a total number of 30 epochs.

It is worth mentioning that all network training is done using SGDM optimizer, unless stated explicitly otherwise. The mini-batch size is 128 samples per iteration. Shuffling is done randomly every epoch to change the order of the image samples during training in order to minimize overfitting and decrease the probability of getting stuck in local minima [24]. The default regularization factor value (0.0001) is used until its optimum value is determined in different configurations. The default momentum value of 0.9 is used whenever the SGDM optimizer is used.

This experiment resulted in 95.06% classification accuracy. For instance, when the ResNet18 model has been modified to accept 28×28 gray images without scaling the input images and without image augmentation we got a relatively low accuracy of 94.81%, which is higher than the reported result (94.44%) in [9], sure because of differences in the training hyperparameters. Hence, the introduced enhancement (0.52%) in the accuracy w.r.t. [9] is due to using different training hyperparameters and scaling up the sizes of the images from 28×28 to 224×224, which is our first step in enhancing the ResNet model's accuracy.

4.2.2 Testing GoogLeNet Model

This model accepts images with the same characteristics as ResNet18, and the images are resized and prepared in the same way mentioned in the previous subsection. Also, the last two layers are replaced with new ones to support 26 classes instead of 1000 classes.

The SGDM optimizer is also used, just as in the previous experiment, but with a base learning rate of 0.001. This experiment resulted in 95.37% classification accuracy.

4.2.3 Testing the Effects of the Selected Pre-Processing and Image Augmentation Methods

In this subsection, the previous two experiments with ResNet18 and GoogLeNet are performed in different situations with the original networks without loading the pre-trained parameters using gray input images (or 1-channel images). In the case of GoogLeNet, Adaptive Moment Estimation (ADAM) learning algorithm [25] is used with a base learning rate of 0.0001 in case of 1-channel images because of convergence problems. In addition, the images are inverted. The effect of extra blank padding around the original character images (as stated in the previous section, where the character images are padded with 6-pixels in the four sides to get a 40×40 pixels image size instead of 28×28 pixels) is also tested in different experiments. Hence, different experiments are performed with the 1-channel version of each model based on the two cases of character sizes (28×28 pixels and 40×40 pixels character sizes). Resizing to 224×224 pixels is done after the padding operation to fit the model input characteristics using bi-cubic interpolation as mentioned before. Rotation and shear augmentation is also tested by generating images after rotation with a random angle in the range of $[-5^\circ\ +5^\circ]$ and shear in both X and Y directions within the range $[-4^\circ\ +4^\circ]$. The results are shown in Tab. 1 for the different cases including the training times in the format of (hours:min:sec). It is important to note that the training times can decrease more than indicated if we prevent accuracy-and-loss plotting. Also, if we consider early stopping, 24 epochs may be enough to get acceptable results but we decided to wait until the training accuracy is considered stable.

Table 1: The accuracies of the single and 3-channel versions of the GoogLeNet and ResNet18 when applied to the EMNIST Letters dataset section under different blank-padding and augmentation conditions

Category	3-channel-images with pre-trained models		1-channel-images with fresh models			
Character size	28×28 pixels		28×28 pixels		40×40 pixels	
Augmentation	None	Rotation $[-5^\circ\ 5^\circ]$ & Shear in X and Y $[-4^\circ\ 4^\circ]$	None	Rotation $[-5^\circ\ 5^\circ]$ & Shear in X and Y $[-4^\circ\ 4^\circ]$	None	Rotation $[-5^\circ\ 5^\circ]$ & Shear in X and Y $[-4^\circ\ 4^\circ]$
ResNet18						
Accuracy (%)	95.06	95.41	95.14	95.35	95.38	95.50
Train-time	04:59:21	05:19:25	03:47:36	04:16:34	04:00:33	04:15:16
GoogLeNet						
Accuracy (%)	95.37	95.54	95.53	95.56	95.45	95.63
Train-time	08:02:27	08:05:19	08:34:25	8:30:48	08:37:02	10:04:40

The experiments are conducted on a desktop computer that is equipped with a Windows 10 Pro operating system, 16 GB random-access memory (RAM), Intel core i7-8700K CPU@3.70 GHz and a graphical accelerated processing unit (GPU) of NVIDIA GeForce GTX 1080 Ti with 11 GB RAM. All experiments are programmed using MATLAB 2019A, and some of the later experiments are programmed with MATLAB 2019B. The main difference between the two versions that affects our experiments lies in the available normalization techniques for the input image layer. In version 2019A there is only zero-mean normalization while in version 2019B there is ZSCORE normalization where each zero mean value is divided by the standard deviation. Most of the experiments are done with the ZSCORE normalization when implemented on the 2019B version. To compensate for this shortage in the 2019A version, we added a batch normalization layer immediately after the input image layer, which stabilizes the results of the experiments by minimizing the variance of the final accuracies when an experiment is repeated many times.

From Tab. 1, it appears that GoogLeNet gives higher accuracies than ResNet18 in all cases, but it takes 2.5 to 2.9 times the ResNet18 training time.

Also, we notice that the cases of 40×40 pixels in the 1-channel versions with rotation–and-shear augmentation have the highest results in both models. An early conclusion is that GoogLeNet models, in all cases except the 28×28 pixels without augmentation case, give higher accuracies than the highest recorded value (95.44%) in the recent survey [3]. Also, the 1-channel version of GoogLeNet with rotation augmentation gives a higher accuracy than the highest state-of-the-art result (95.58%). The ResNet18 model gives a higher accuracy than 95.44% in the case of blank pixel padding (40 × 40 pixels) and rotation-and-shear augmentation. From the above table, we conclude that image augmentation using rotation and shear combined with blank pixel padding increased the accuracy by 0.36% and 0.10% for the 1-channel versions of ResNet18 and GoogLeNet models, respectively.

4.3 Changing the Number of Network Building Blocks

In this subsection, the number of residual blocks are varied in the 1-channel version of ResNet18, and the accuracy is recorded in each case. The same is done for the GoogLeNet version where the number of inception blocks are varied, and the corresponding accuracy is recorded in each case. Rotation augmentation in the angle range [-5° $+5^\circ$] and shear augmentation in both X and Y directions in the range [-4° 4°] are done in all experiments for the 40 × 40 pixels blank-extended version of the dataset. The base learning rate is changed to 0.0001 in the GoogLeNet experiments in the cases from 8 to 11 blocks, because of convergence problems. It is important to note that we use the terminology of the MATLAB documentation, where each type of important transformation or operation done separately is considered as a layer.

Tab. 2 displays the number of blocks (residual or inception) and the corresponding number of used layers with the resultant network classification accuracy of the EMNIST Letters test dataset.

Table 2: Classification accuracy *vs.* the number of inception or residual building blocks when the modified (ResNet18 or GoogLeNet) model is tested on the EMNIST Letters dataset section (maximum values are bold)

GoogLeNet				ResNet18			
Number of inception blocks	Number of layers	Base learning rate	Classification accuracy	Number of residual blocks	Number of layers	Base learning rate	Classification accuracy
2	44	0.001	94.63	2	26	0.1	93.61
3	59	0.001	95.22	3	32	0.1	93.50
4	73	0.001	95.33	4	41	0.1	95.32
5	87	0.001	95.48	5	48	0.1	95.53
6	101	0.001	95.55	6	55	0.1	95.68
7	115	0.001	95.58	7	64	0.1	**95.75**
8	130	0.0001	95.61	8	71	0.1	95.50
9	144	0.0001	95.63	9	80	0.1	95.40
10	158	0.0001	**95.73**				
11	172	0.0001	95.66				

From Tab. 2, we notice the following:

- The accuracy of both models increases with the increase of the number of blocks until a certain threshold, 10 inception blocks for the GoogLeNet based model and 7 residual blocks for the ResNet18 based model. After this threshold, the accuracy starts decreasing in both models with the increase of the number of blocks.
- While the threshold of the residual network is lower than the number of the residual blocks in the original network (8 in ResNet18), the threshold in the inception network is higher than the number of inception blocks of the original network (9 in GoogLeNet).

Hence, the residual network with 6 or 7 blocks is selected as the solution because it has simpler architecture with slightly higher accuracy and lower training time, and surely it will have lower testing time. In the following subsections, fine-tuning will be performed to increase the accuracy of the residual solution.

4.4 Fine-Tuning of the Residual Solution

In this section, fine-tuning will be carried out by adding a drop-out layer with a proper drop-out rate (or probability) (DrOPr). Then the receptive field of the first CF will be varied to select the best size. Also, the hyper parameter, L2 Norm regularization factor (L2RF) will be tested to select the best value, and finally the rotation angle augmentation range will be experimented to determine the best range.

For instance, many other experiments have been performed to test the effects of other structural and parameter variations in the selected architecture, but most of them didn't result in sensed improvements in the performance of the model such as changing the size of the filters inside the residual blocks to sizes other than 3×3 or changing the activation functions types to types other than the ReLU. Hence, we mention only those structural and parameter changes that led to significant improvements in the model accuracy. Since we are facing a challenging problem, we decide to include a modification in our model if it increases the average accuracy by 0.01% or more.

4.4.1 Testing the Effect of the Addition of a Drop-Out Layer Before the Fully Connected Layer

In this subsection, we test the effect of adding a drop-out layer before the fully connected one in four cases of the residual solution starting from the 5-block version until the 8-block version of the residual model and select the best DrOPr value (s) based on the attained accuracies in each case. We selected four cases to test because this modification affects the number of blocks to retain in our model in conjunction with the insertion of a drop-out layer with proper DrOPr. Since the 8-block version has the same layout as the original model of ResNet18 (with a 1-channel image input), hence we display only the layout of the 7-block residual network in Fig. 5 with the addition of a global variable named as Number of Filters (NF). The NF variable represents the number of CFs in the first residual block that is doubled every time down-sampling with a factor of 2 is performed in the subsequent blocks. The 6-block network is the resultant network after the removal of the seventh block and changing the average pooling to get the average of 14×14 instead of 7×7 and replacing the fully connected (dense) layer with one having 256×26 instead of 512×26. The 5-block version is obtained in the same way.

Twenty-eight experiments have been performed by adding a drop-out layer before the fully connected layer in the 5-block until the 8-block versions and changing DrOPr from 0.2 to 0.8 to find the best layout with the best DrOPr value. Training using SGDM continues for 24 epochs, with a drop factor of 0.1 every 8 epochs. The results of the average accuracy in each case are shown in Tab. 3.

From Tab. 3, we notice that the drop-out layer enhances the 5-block and 6-block versions (bold values) while it does not enhance the 7-block version. The 8-block version is enhanced slightly, but still has a lower accuracy than intended. Moreover, we can conclude that adding a drop-out layer gives higher accuracy than adding an extra residual block at the end of the residual network (sure after reaching a certain threshold). We also deduce that the best drop-out probability is 0.7 for the 6-block case, which has an enhanced accuracy of 95.81%.

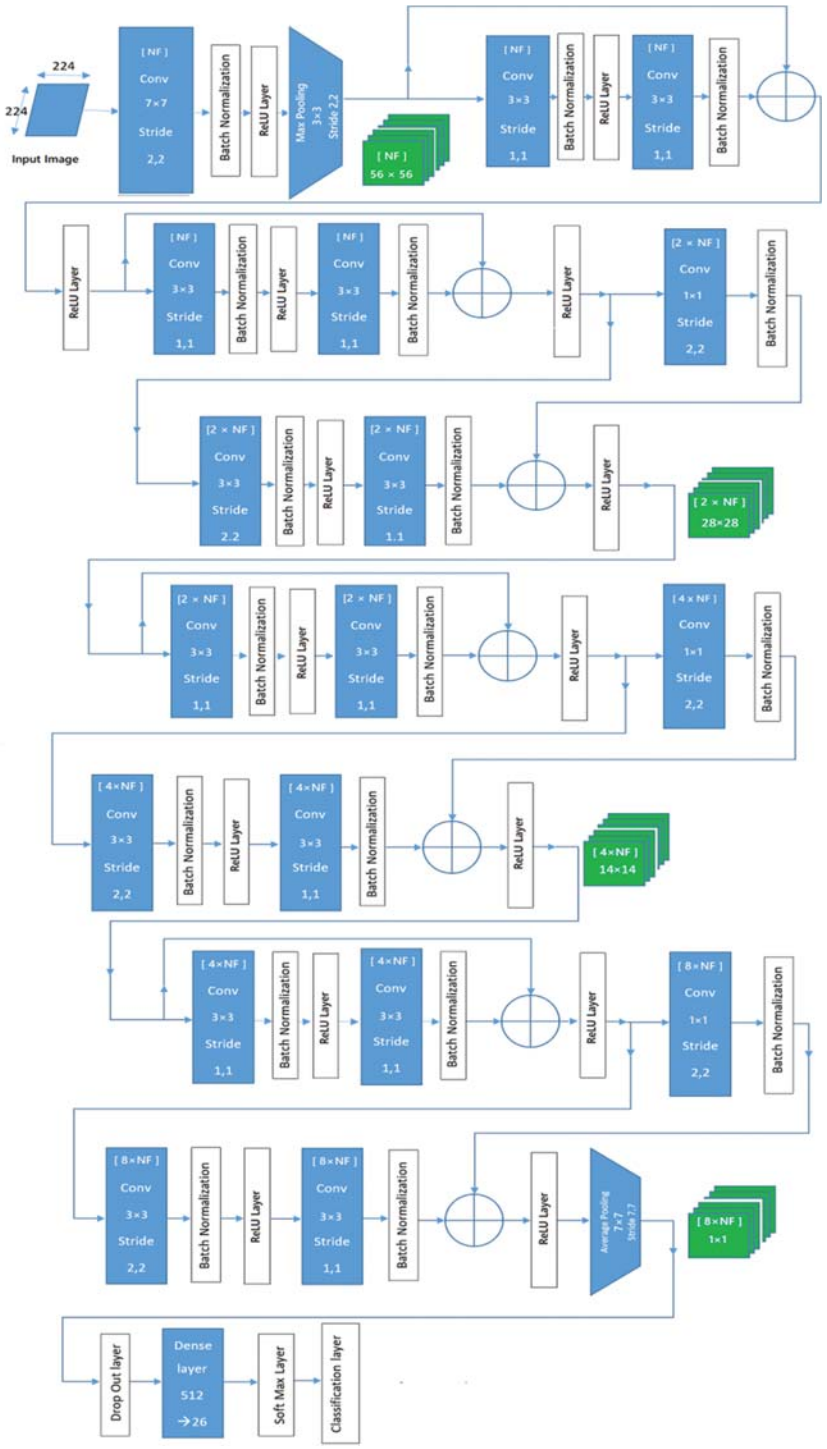
224
224
Input Image
[NF] Conv 7×7 Stride 2,2
Batch Normalization
ReLU Layer
Max Pooling 3×3 Stride 2,2
[NF] 56 × 56
[NF] Conv 3×3 Stride 1,1
[2 × NF] Conv 1×1 Stride 2,2
[2 × NF] Conv 3×3 Stride 2,2
[2 × NF] 28×28
[4×NF] Conv 1×1 Stride 2,2
[4×NF] Conv 3×3 Stride 2,2
[4×NF] 14×14
[8×NF] Conv 1×1 Stride 2,2
[8×NF] Conv 3×3 Stride 1,1
Average Pooling 7×7 Stride 7,7
[8×NF] 1×1
Drop Out layer
Dense layer 512 →26
Soft Max Layer
Classification layer

Figure 5: The 7-block residual CNN network, where the number of CFs is denoted by the number enclosed within brackets in the convolution unit and the resultant number of FMs is also denoted by the number enclosed within brackets in the multi-channel symbols that are not connected to the graph

Table 3: The classification accuracy *vs.* the drop-out probability (DrOPr) in case of 5-block, 6-block, 7-block, and 8-block residual networks when applied to the EMNIST Letters dataset section

DrOPr	0.2	0.3	0.4	0.5	0.6	0.7	0.8
5 Blocks	95.52%	**95.63%**	95.53%	95.51%	95.47%	95.38%	95.39%
6 Blocks	**95.71%**	**95.75%**	**95.74%**	**95.72%**	**95.80%**	**95.81%**	95.65%
7 Blocks	95.60%	95.61%	95.64%	95.67%	95.69%	95.71%	95.61%
8 Blocks	95.48%	**95.60%**	**95.57%**	**95.65%**	**95.63%**	**95.65%**	**95.60%**

Hence, the 7th and 8th blocks will be removed, and we will present further modifications of the 6-block version in the following subsections.

4.4.2 Testing the Effect of the Size of the First CF

The first CF in ResNet18 has a 7×7 size with a stride of 2 in both directions. In this experiment the filter size will be varied from 5×5 to 13×13 using odd values only, and the corresponding average accuracy will be recorded in each case given the same conditions in the previous experiment with a drop-out probability of 0.7 in the 6-block version.

The first max-pooling filter is replaced by an average pooling filter to get more stable results because the max-pooling filter increases the character stroke thickness, while the average pooling filter smooths the edges of the character strokes without increasing their thickness. Tab. 4 shows the corresponding accuracy found for each filter size when using $[-5^\circ\ 5^\circ]$ rotation and $[-4^\circ\ 4^\circ]$ shear augmentation ranges.

Table 4: The classification accuracy *vs.* the first CF size in case of the 6-block residual network

Filter size	5×5	7×7	9×9	11×11	13×13
Average accuracy (%)	95.77	95.81	95.82	95.83	95.80

From this experiment, we can conclude that increasing the receptive field of the first filter to 9×9 or 11×11 gives higher accuracies than the original 7×7 in the case of the EMNIST Letters dataset.

Hence, we selected the two models with the 9×9 and 11×11 filter sizes to be enhanced in the following modification.

4.4.3 Testing the Effect of the L2 Norm Regularization Factor Value (L2RF)

The purpose of the L2 Norm regularization is to reduce overfitting by controlling change in the network (filters) weights during the optimization process through the regularization factor [26].

In this subsection, L2RF is changed from 0.0001 to 0.0009 and the resultant accuracy is recorded in each case for the 9×9 and 11×11 filters.

Tab. 5 lists the results of eighteen experiments done under the best conditions of the previous experiments (having a drop-out layer with probability 0.7, $[-5^\circ\ 5^\circ]$ rotation and $[-4^\circ\ 4^\circ]$ shear augmentation ranges and first CF size 9×9 or 11×11) but with L2RF varied from 0.0001 up to 0.0009.

Table 5: The classification accuracy *vs.* the L2 regularization factor value

L2RF/Filter size	0.0001	0.0002	0.0003	0.0004	0.0005	0.0006	0.0007	0.0008	0.0009
9×9	95.82%	95.84%	95.86%	95.88%	95.84%	95.82%	95.80%	95.79%	95.77%
11×11	95.83%	95.85%	95.87%	95.89%	95.86%	95.84%	95.83%	95.82%	95.81%

From Tab. 5, it can be concluded that the two values (0.0003 and 0.0004) of L2RF enhance the resulting average classification accuracy by about 0.05%, or in other words increase the number of correctly recognized characters by approximately 10 characters. Hence we select the value of 0.0004, which gave the highest average accuracy for the 11×11 filter case.

4.4.4 Testing the Effects of Changing the Rotation Angle Augmentation Range

In this experiment we tested the 9×9 and 11×11 filters with different rotation angle augmentation ranges starting from [$-5°$ to $5°$] up to [$-25°$ to $25°$] with a step of ($-5°$, $5°$) degrees where the resultant accuracy in each case is shown in Tab. 6.

From Tab. 6, it appears that the best rotation angle augmentation range is [$-5°$ $5°$]. The 6-block residual solution with 11×11 first CF and drop-out layer is referred to as Res6BF11.

Table 6: The classification accuracy *vs.* the rotation angle augmentation range

Angle-range	No augmentation	$-5°$:$5°$	$-10°$:$10°$	$-15°$:$15°$	$-20°$:$20°$	$-25°$:$25°$
Filter size						
9×9	95.83%	95.88%	95.80%	95.75%	95.65%	95.40%
11×11	95.84%	95.89%	95.81%	95.77%	95.67%	95.43%

4.4.5 Testing the Effect of Changing the Final Addition Layer into a Depth Concatenation Layer

Although changing the addition layer into a concatenation layer is a structural variation that should be done first with other structural variations, it has been delayed because it is a foreign variation where the modified model will no longer be a pure residual network. It is known that information is propagated through the residual network from block to block by keeping the original information and the mapped one in each level throughout the addition operation such that the output of the final block can be considered as the sum of all the outputs of the previous blocks (if we neglect the non-linear effects of the ReLU layers) as in (3). If we replace the final addition layer with a depth concatenation layer, we present the information of the penultimate residual block and the mapped information out of the final block in separate channels (FMs) instead of merging them through addition in the same channels (FMs). Although this type of information merging when implemented in the whole network proved superior in the previously developed DenseNet [27] when applied to the ImageNet, it did not show such superiority when tested without modifications on the EMNIST Letters dataset.

To illustrate the difference, we rewrite the recursive formula (3) in case of a Dense block of N modules as follows:

$$\mathrm{DenseOut}(N) = \begin{cases} F_N(\mathrm{DenseOut}(N-1)) \,\|\, \mathrm{DenseOut}(N-1) & N > 0 \\ \mathrm{Input}_0 & N = 0 \end{cases} \tag{4}$$

where the operator '‖' means depth concatenation of the left and right operands.

Although DenseNet is one of the important advanced architectures of the DCNN, we didn't refer to it in the start of this research because its available ready-made models are considered heavyweight when compared with GoogLeNet and ResNet18.

A 1-channel version of DenseNet-BC-121 after being trained on the EMNIST Letters for 30 epochs with 0.1 drop factor every 6 epochs yielded an accuracy of 95.62% on the test set. It took about 111.7 hours of training compared to 4.15 hours for the ResNet18 and 10.04 hours for the GoogleNet that yielded accuracies of 95.50% and 95.63%, respectively as shown in Tab. 1.

This is because unlike the ImageNet, the images of the characters in the EMNIST dataset have single scale and mainly appear as a whole (not partially), and hence no need to give the representations in the earlier layers (or blocks) separate channels in the final layers (or blocks), which will increase the redundant information at the input of the final block.

Hence, we exchanged the addition layer with depth concatenation layer only in the final block and recorded the resulting accuracy after making the different types of parameter tuning, as done before.

Based on many experiments, we found that the best parameters are all the same as given in the previous subsections except the L2RF value, which is changed to 0.0005 to get the highest accuracy.

The resulting output of the final dense block, which is preceded by 5 residual blocks (Dense1Res5Out) can be represented by the following formula:

$$\mathrm{Dense1Res5Out}(N) = F_N(\mathrm{ResOut}(N-1)) \| \mathrm{ResOut}(N-1) \tag{5}$$

where $\mathrm{ResOut}(N-1)$ is defined in (3).

In the general case if there are R residual modules and D dense final modules we call this architecture DenseDResR. This is the reason that we referred to our final model defined in (5) by Dense1Res5.

After repeating the training of the new network architecture fifteen times and recording the accuracies of the generated models on the Letters test set, we got the accuracy values listed in Tab. 7.

Table 7: The accuracy of the fifteen trained models when each is used to classify the letters test set

Model#	1	2	3	4	5
Accuracy (%)	95.88	95.93	95.87	95.94	95.94
Model#	6	7	8	9	10
Accuracy (%)	95.90	95.91	95.94	95.92	95.90
Model#	11	12	13	14	15
Accuracy (%)	95.87	95.97	95.92	95.88	95.90

Hence, we conclude that the accuracy of the proposed model has a mean value of 95.91% with a standard deviation (STD) of 0.0003. For instance, we have replaced the last two blocks ($D = 2$, $R = 4$) with two depth concatenation blocks, but we didn't gain the same improvement where the resultant accuracy was about 95.80%. We have also changed all addition layers to depth concatenation layers ($D = 6$, $R = 0$), which resulted in an average accuracy of 95.75%.

4.5 Aggregation of Different Instances of the Proposed Model

By selecting m models randomly out of the 15 models generated in the previous experiment we estimate the accuracy of the aggregated models based on the average of the class probabilities of the m models.

By performing 1000 random permutations for each value of m starting from 2 up to 9 and calculating the mean and standard deviation in each case, we got the results shown in Tab. 8 and plotted in Fig. 6.

Since our goal is to maximize the mean and minimize the STD of the classification accuracy without increasing the training and testing time too much, we will allow m to be increased as long as there are sensed improvements in the mean or in the STD to some extent.

From Tab. 8 and Fig. 6, we observe that the mean reaches its maximum at $m = 5$ (95.98%) and remains constant after that, while the STD has a value of 2.65×10^{-4} at $m = 5$ and decreases slightly after that, only decreasing by a factor of 0.79 when m changes from 5 to 9, so we decided to select an ensemble of 5 models to be used in the four cases in this research (full and reduced training sets of the Letters and Balanced dataset sections). Using an ensemble of 9 models is a good option because it will decrease STD by a factor of 0.79 but will increase both the training and testing times linearly by a factor of 9/5 and also increase the required memory for storing the probability matrices by the same factor without gaining improvement in the mean of the accuracy.

Table 8: The mean and standard deviation of the classification accuracy of the Letters test set *vs.* the number of aggregated models (m)

m	2	3	4	5	6	7	8	9
Mean (%)	95.96	95.97	95.97	**95.98**	95.98	95.98	95.98	95.98
STD	0.00036	0.00033	0.0003	**0.000265**	0.000262	0.00024	0.00023	0.00021

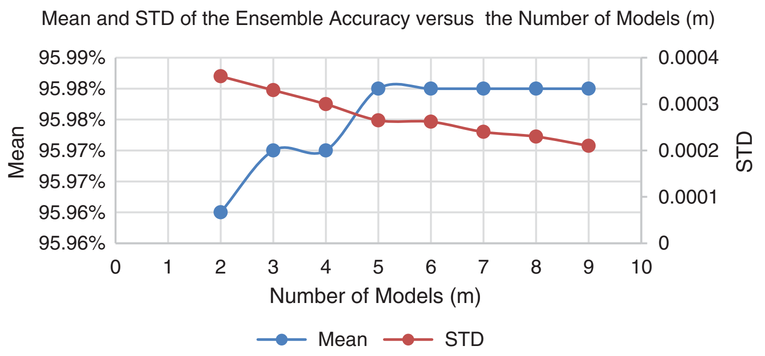

Figure 6: Mean and STD of the ensemble accuracy *vs.* the number of models (m)

5 The Final Experiments

In this section, the final experiments are performed on the Balanced dataset section, and the results of both dataset sections are summarized when using a single model or an ensemble under the best conditions of the proposed models (Res6BF11 and Dense1Res5), which have a drop-out probability of 0.7 and a regularization factor of 0.0004 for Res6BF11 and 0.0005 for Dense1Res5 and using [−5° 5°] and [−4° 4°] ranges for rotation and shear augmentations, respectively.

Then, the same experiments are repeated on both dataset sections but with training on a reduced dataset having 200 samples per class instead of 4800 samples per class in the Letters dataset and 2400 samples per class in the Balanced dataset respectively.

5.1 Experiments on the Balanced Dataset Section

In this subsection, the proposed models are applied to the Balanced dataset section, but an extra experiment will be carried out to test the effect of doubling the number of CFs (NF = 128) in all the residual blocks of the model.

When we applied the Res6BF11 and Dense1Res5 models to the Balanced dataset section, we got an average accuracy of 90.75% and 90.82%, respectively but when we doubled the number of CFs (NF = 128), we got an average accuracy of 90.90% and 91.00%, respectively. This enhancement gained by doubling the number of CFs was not observed on the Letters dataset section. This might be due to having more classes in the Balanced section (47) than in the Letters section (26). Finally, the training of the model Dense1Res5 (NF = 128) was performed fifteen times on the Balanced dataset, and the generated versions were used to classify the test set. Tab. 9 displays the results of the fifteen experiments.

Table 9: The accuracy of fifteen versions of Dense1Res5 (NF = 128) when each is used to classify the balanced test set

Model#	1	2	3	4	5
Accuracy (%)	91.04	90.91	91.01	90.89	90.97
Model#	6	7	8	9	10
Accuracy (%)	90.97	91.01	90.96	90.95	91.08
Model#	11	12	13	14	15
Accuracy (%)	91.01	91.06	91.11	91.01	90.96

From Tab. 9, the mean is 91.00% and STD is 5.9×10^{-4}.

Tab. 10 gives the results of the aggregation of m models out of the fifteen versions. From Tab. 10, the mean when using 5 models is 91.06% and STD is 4.5×10^{-4}.

Table 10: The mean and standard deviation of the classification accuracy of the balanced test set *vs.* the number of aggregated models (m)

m	2	3	4	5	6	7	8	9
Mean (%)	91.04	91.05	91.06	91.06	91.06	91.07	91.07	91.07
STD	0.00053	0.00051	0.00047	0.00045	0.00044	0.00041	0.00039	0.00036

5.2 Testing the Proposed Models on the Reduced Training Dataset of the EMNIST Letters Section

We trained the Res6BF11 and Dense1Res5 models on the reduced training letters dataset section but with rotation angle augmentation range of [-10° 10°] and initial learning rate of 0.1, which is divided by 0.1 every 17 epochs for a total of 52 epochs. Then, we used the generated models to classify the test dataset section. We got an average accuracy of 93.42% and 93.56% for the Res6BF11 and Dense1Res5 models (with NF = 64), respectively.

For Res6BF11 and Dense1Res5 models with NF = 128, we used a mini-batch size of 64 and got an average accuracy of 93.77% and 93.82%, respectively.

Tab. 11 shows the resultant accuracies of fifteen experiments done with the Dense1Res5 model (NF = 128), which have a mean value of 93.82% (STD = 6.0×10^{-4}). When we used an ensemble of 5 models out of the Dense1Res5 fifteen versions, we got a mean accuracy of 93.94% (STD = 3.78×10^{-4}).

Table 11: The accuracy of fifteen versions of Dense1Res5 model (NF = 128) trained on the reduced Letters set when each is used to classify the Letters test set

Model#	1	2	3	4	5
Accuracy (%)	93.76	93.80	93.74	93.75	93.77
Model#	6	7	8	9	10
Accuracy (%)	93.80	93.87	93.81	93.94	93.84
Model#	11	12	13	14	15
Accuracy (%)	93.79	93.82	93.85	93.82	93.94

5.3 Testing the Proposed Models on the Reduced Training Dataset of the EMNIST Balanced Section

We trained the Res6BF11 and Dense1Res5 models on the reduced training dataset section using a rotation angle augmentation range of [-10 $+10$] and NF = 128 (with mini-batch size of 64). Then, we used the generated models to classify the test dataset section. We got an average accuracy of 88.49% and 88.69% for Res6BF11 and Dense1Res5 models, respectively. Tab. 12 shows the resultant accuracies of fifteen experiments done with the Dense1Res5 (NF = 128) model which has a mean value of 88.69% (STD = 7.8×10^{-4}).

Table 12: The accuracy of fifteen versions of Dense1Res5 model (NF = 128) trained on the reduced balanced set when each is used to classify the Balanced test set

Model#	1	2	3	4	5
Accuracy (%)	88.84	88.65	88.65	88.78	88.79
Model#	6	7	8	9	10
Accuracy (%)	88.67	88.61	88.71	88.64	88.57
Model#	11	12	13	14	15
Accuracy (%)	88.80	88.68	88.61	88.66	88.71

When we used an ensemble of 5 models of the fifteen Dense1Res5 versions, we got a mean accuracy of 88.83% (STD = 4.7×10^{-4}).

6 Comparisons with State-of-the-Art Results

Tab. 13 summarizes the results for the full data set and the reduced one giving the accuracies of the proposed solutions (Res6BF11 and Dense1Res5 for NF = 64 and NF = 128) in conjunction with the base classifier (OPIUM), the basic ResNet model [9], and the highest results (bold style) of the state-of-the-art techniques mentioned in the literature review. Our ensemble accuracies (bold underlined) are calculated only for the best single model solution (bold italics) in each case.

Table 13: Classification accuracy of the proposed models and the state-of-the-art systems for the full and reduced training sets of the Letters and Balanced dataset sections

Model	Letters (Full) (%)	Balanced (Full) (%)	Letters (reduced) (%)	Balanced (reduced) (%)
OPIUM (Base) [1]	85.15	78.02		
Basic ResNet [9]	94.44			
Genetic DCNN [9]	**95.58**			
TextCaps [12]	95.36	90.46	**92.79**	**87.82**
DCNN (6 conv + 2 dense) [15]		**90.59**		
Res6BF11 (NF = 64)	95.89	90.75	93.42	88.37
Dense1Res5 (NF = 64)	***95.91***	90.82	*93.56*	88.50
Res6BF11 (NF = 128)	95.87	90.90	*93.77*	88.49
Dense1Res5 (NF = 128)	95.88	***91.00***	***93.82***	***88.69***
Ensemble of 5 Dense1Res5 (NF = 64)	**95.98**			
Ensemble of 5 Dense1Res5 (NF = 128)		**91.06**	**93.94**	**88.83**

Based on Tab. 13 and considering the results of the Letters dataset, we have increased the accuracy from the highest published value 95.58% to 95.98%, which is equivalent to decreasing the error rate by 9% using five Dense1Res5 (NF = 64) models each having about 1.69 M parameters.

Considering the Balanced dataset, we have increased the accuracy from the highest published value 90.59% to 91.06% which is equivalent to decreasing the error rate by 5% using five Dense1Res5 (NF = 128) models each having about 6.7 M parameters.

Considering the reduced training set of the Letters and Balanced datasets, we have increased the accuracies by 1.15% and 1.01%, respectively, over the TextCaps model published results, which is equivalent to decreasing the error rate by 15.95% and 8.29%, respectively.

Based on the testing done by [9] using the basic architecture of ResNet [18] on the Letters dataset, we have increased the accuracy of the basic ResNet architecture from 94.44% to 95.91% using a single instance of the Dense1Res5 (NF = 64) model. In the case of using an ensemble of 5 Dense1Res5 (NF = 64) models, the enhancement will be clearer with a 1.54% increase in the achieved accuracy (95.98%), which corresponds to a 27.7% improvement in the error rate. This improvement in the basic architecture is our significant contribution due to the distinguished optimization of the residual solution.

Although the enhancements due to the replacement of the addition layer with the depth concatenation layer appear to be very small in the results of the full Letters dataset test (0.01% (NF = 128) and 0.02% (NF = 64)), it is relatively higher in the results of the full Balanced dataset test (0.07% (NF = 64) and 0.10% (NF = 128)). The situation is better in the case of the reduced training sets with an enhancement of 0.14% (NF = 64) and 0.05% (NF = 128) for the Letters dataset and enhancement of 0.13% (NF = 64) and 0.20% for the Balanced dataset. These relatively small enhancements will be appreciated if we consider the achievements done in the last three years where, as stated in the literature review, an enhancement of 0.14% had been achieved after approximately 30 months in the classification accuracy of the Letters dataset. Another convincing example is the use of a committee of 20 CNNs to achieve an enhancement of 0.14% in the classification accuracy of the EMINST-Letters dataset [10]. It should be noted that the greater the accuracy, the more difficult the improvement will be. Therefore, to be fair we have to calculate the percentage of improvement in the error rate when comparing different improvements at different accuracy levels. Thus, the 0.14% accuracy enhancement achieved in 30 months (in the classification accuracy of the Letters dataset) corresponds to a 3.07% improvement in the error rate, which is one-third of our improvement in the error rate (9%).

When we compare our work with the base model as done in most of the mentioned researches, we conclude that the accuracy has been increased by 10.83% and 13.04% for the Letters and Balanced datasets, respectively.

To allow reproducibility of the results and prevent any missing information in the proposed model, the MATLAB code that defines the layer graph of the Dense1Res5 model is listed in Appendix A.

7 Conclusions and Future Work

In this research, the recognition accuracy of the EMNIST dataset using deep residual CNN has been improved by the following solution dimensions:

- Data dimension: the dataset images have been adapted with the residual CNN using the proper zero paddings and scaling and then proper rotation and shear angle augmentation have been performed to increase the trained model's generalization.
- Structure dimension: the architecture of the residual solution has been enhanced by the selection of the optimum number of residual blocks, the optimum size of the receptive field of the first CF, the replacement of the first max-pooling layer with an average pooling layer, and the addition of the drop-out layer before the fully connected layer. A novel enhancement of the architecture has been achieved by replacing the final addition layer with a depth concatenation layer. Furthermore, doubling up the number of filters in all convolutional layers has brought enhancements in the subsequent accuracy, particularly in the Balanced dataset.
- Training hyperparameter dimension: the hyperparameters (learning rate, L2 regularization factor, mini-batch size, and the number of epochs in the training process) have been optimized for each proposed model in the four dataset cases (full Letters and Balanced datasets and their reduced versions).
- Aggregation of several models: by averaging the class probabilities of 5 versions of each proposed model the overall accuracy has been improved.

The improvements done to the residual solution starting with image scaling and data augmentation and ending with the hyperparameter optimization, resulted in a 26.43% improvement in the error rate in the classification of the EMNIST letters test set, relative to the basic residual architecture.

After using an ensemble of 5 versions of each proposed model, the improvements in the error rates relative to the stat-of-the-arts error rates become approximately 9%/5% for the full

Letters/Balanced datasets and 16%/8% for the reduced training sets of the Letters/Balanced datasets, respectively.

The mentioned improvements in the residual DCNN model allowed the modified model to outperform the state-of-the-art models such as the TextCaps model in the classification of the handwritten characters when using either a huge training dataset or a reduced one.

On the basis of the experiments conducted in this work, we have also reached the following general conclusions for ResNet18 or similar architectures when being utilized in handwritten character image classification tasks:

- There is an optimal resolution of the input layer, in which the original image takes an optimal proportion that gives the maximum accuracy for a certain network architecture.
- Using rotation angle augmentation range of $[-5^\circ\ +5^\circ]$ increases the final accuracy in the case of the full sets, that is doubled to $[-10^\circ\ +10^\circ]$ in the case of the reduced sets, to fill the gabs left due to unused samples by using a higher number of training epochs with slower learning rate changes.
- There is an optimal number (Nopt) of residual blocks after which the accuracy does not improve or slightly decreases.
- Using (Nopt-1) blocks with a drop-out layer results in higher accuracy than using (Nopt) blocks even if a drop-out layer is used.
- Finalizing the residual net with a depth concatenated block generates a more accurate model than using a pure residual network when dealing with handwritten characters that have an approximately single scale and appear mostly in full shapes.

With regards to the architectures of DCNN, we have come up with a novel architecture which combined the residual network with the dense network in a sequential manner and offered a recursive definition for the original architectures and the new one, termed DenseDResR that could be evaluated in the future works with varying values of D and R on other dataset variants. We also gave an explanation why depth concatenation was beneficial in the final block only in the case of the EMNIST dataset based on the existence of full shapes of the characters in almost one scale in all the data set images. Other problems which give a decision of the presence of a class based on different scales of its shape or the partial existence of its shape such as ImageNet are expected to have a large D and small R where the lower abstractions of a shape (part of a circle for example) affect the final decision of the network and hence take a proportion in the final feature representation or abstraction. We think that using a large D irrelevant to the classification problem under consideration will introduce redundancy in the final and possibly in the intermediate layers of a network that will increase the memory requirements and the training time without having significant improvements in the resultant accuracy.

Acknowledgement: The authors acknowledge Prof. Abdullah Basuhail who supervises the Thesis of Mr. Hadi and did the proofreading of this paper.

Funding Statement: The authors received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] G. Cohen, S. Afshar, J. Tapson and A. Van Schaik, “EMNIST: Extending MNIST to handwritten letters,” in *Proc. IJCNN 2017*, Anchorage, AK, USA, pp. 2921–2926, 2017.

[2] Y. LeCun, C. Cortes and C. BURGES, “The MNIST database of handwritten digits,” 2012. [Online]. Available: http://yann.lecun.com/exdb/mnist/.

[3] A. Baldominos, Y. Saez and P. Isasi, “A survey of handwritten character recognition with MNIST and EMNIST,” *Applied Sciences*, vol. 9, no. 15, pp. 31692019.

[4] A. van Schaik and J. Tapson, “Online and adaptive pseudoinverse solutions for ELM weights,” *Neurocomputing*, vol. 149, pp. 233–238, 2015.

[5] Y. Peng and H. Yin, “Markov random field based convolutional neural networks for image classification,” *Proc. IDEAL*, vol. 10585, pp. 387–396, 2017.

[6] P. Ghadekar, S. Ingole and D. Sonone, “Handwritten digit and letter recognition using hybrid DWT-DCT with KNN and SVM classifier,” in *Proc. ICCUBEA*, Pune, India, pp. 1–6, 2018.

[7] R. Wiyatno and J. Orchard, “Style memory: Making a classifier network generative,” in *Proc. (ICCI*CC)*, Berkeley, CA, USA, pp. 16–21, 2018.

[8] E. Dufourq and B. A. Bassett, “Eden: Evolutionary deep networks for efficient machine learning,” in *Proc. (PRASA-RobMech)*, Bloemfontein, South Africa, pp. 110–115, 2017.

[9] B. Ma, X. Li, Y. Xia and Y. Zhang, “Autonomous deep learning: A genetic DCNN designer for image classification,” *Neurocomputing*, vol. 379, pp. 152–161, 2020.

[10] A. Baldominos, Y. Saez and P. Isasi, “Hybridizing evolutionary computation and deep neural networks: An approach to handwriting recognition using committees and transfer learning,” *Complexity*, vol. 2019, pp. 1–16, 2019.

[11] P. Cavalin and L. Oliveira, “Confusion matrix-based building of hierarchical classification,” in *Proc. Iberoamerican Congress on Pattern Recognition*, CIARP, Madrid, Spain, pp. 271–278, 2018.

[12] V. Jayasundara, S. Jayasekara, H. Jayasekara, J. Rajasegaran, S. Seneviratne *et al.,* “Textcaps: Handwritten character recognition with very small datasets,” in *Proc. WACV*, Waikoloa Village, HI, USA, pp. 254–262, 2019.

[13] S. Sabour, N. Frosst and G. E. Hinton, “Dynamic routing between capsules,” in *Advances in Neural Information Processing Systems, NIPS 2017*. Long Beach, CA, USA: NEURAL INFORMATION PROCESSING SYSTEMS (NIPS), pp. 3856–3866, 2017.

[14] K. S. Younis, “Arabic handwritten character recognition based on deep convolutional neural networks,” *Jordanian Journal of Computers and Information Technology*, vol. 3, no. 3, pp. 186–200, 2017.

[15] A. Shawon, M. J. U. Rahman, F. Mahmud and M. A. Zaman, “Bangla handwritten digit recognition using deep CNN for large and unbiased dataset,” in *Proc. ICBSLP*, Sylhet, Bangladesh, pp. 1–6, 2018.

[16] A. Khan, A. Sohail, U. Zahoora and A. S. Qureshi, “A survey of the recent architectures of deep convolutional neural networks,” *Artificial Intelligence Review*, vol. 53, no. 8, pp. 5455–5516, 2020.

[17] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed *et al.,* “Going deeper with convolutions,” in *Proc. CVPR*, Boston, MA, USA, pp. 1–9, 2015.

[18] K. He, X. Zhang, S. Ren and J. Sun, “Deep residual learning for image recognition,” in *Proc. CVPR*, Las Vegas, NV, USA, pp. 770–778, 2016.

[19] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *arXiv preprint*, arXiv:1502.03167, 2015.

[20] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[21] X. Lin, Y. L. Ma, L. Z. Ma and R. L. Zhang, “A survey for image resizing,” *Journal of Zhejiang University SCIENCE C*, vol. 15, no. 9, pp. 697–716, 2014.

[22] A. Poznanski and L. Wolf, “CNN-N-Gram for handwriting word recognition,” in *Proc. CVPR*, Las Vegas, NV, USA, pp. 2305–2314, 2016.

[23] Y. Yan, T. Yang, Z. Li, Q. Lin and Y. Yang, “A unified analysis of stochastic momentum methods for deep learning,” *arXiv preprint*, arXiv:1808.07576, 2018.

[24] Z. L. Ke, H. Y. Cheng and C. L. Yang, “LIRS: Enabling efficient machine learning on NVM-based storage via a lightweight implementation of random shuffling,” *arXiv preprint*, arXiv:1810.04509, 2018.

[25] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint*, arXiv:1412.6980, 2014.

[26] P. Murugan and S. Durairaj, “Regularization and optimization strategies in deep convolutional neural network,” *arXiv preprint*, arXiv:1712.04711, 2017.

[27] G. Huang, Z. Liu, L. Van Der Maaten and K. Q. Weinberger, “Densely connected convolutional networks,” in *Proc. CVPR*, Honolulu, HI, USA, pp. 4700–4708, 2017.

Appendix A. Dense1Res5 Lgraph Creation Code

```
% Script for creating the layers for a deep-learning network with: Number of layers: 57, Number
of connections: 62
% Running this script will create the layers in the workspace variable lgraph.
% Create the layer graph variable to contain the network’s layers.
lgraph = layerGraph( );
NF = 64; % define the base number of filters
% Add the Layer Branches, Add the branches of the network to the layer graph, Each branch is
a linear array of layers.
tempLayers = [
imageInputLayer([224,224,1],“Name”, “data”, “Normalization”, “none”)
batchNormalizationLayer(“Name”, “bn_conv1_1_2”)
convolution2dLayer([11,11], NF, “Name”, “conv1_1”, “BiasLearnRateFactor”, 0, “Padding”, [5 5
5 5], “Stride”, [2 2])
batchNormalizationLayer(“Name”, “bn_conv1_1_1”)
reluLayer(“Name”, “conv1_relu”)
averagePooling2dLayer([3,3], “Name”, “avgpool2d”, “Padding”, [1 1 1 1], “Stride”, [2 2])];
lgraph = addLayers(lgraph,tempLayers);
tempLayers = [
convolution2dLayer([3,3], NF, “Name”, “res2a_branch2a”, “BiasLearnRateFactor”, 0, “Padding”,
[1 1 1 1])
batchNormalizationLayer(“Name”, “bn2a_branch2a”)
reluLayer(“Name”, “res2a_branch2a_relu”)
convolution2dLayer([3,3], NF, “Name”, “res2a_branch2b”, “BiasLearnRateFactor”, 0, “Padding”,
[1 1 1 1])
batchNormalizationLayer(“Name”, “bn2a_branch2b”)];
lgraph = addLayers(lgraph, tempLayers);
tempLayers = [additionLayer(2, “Name”, “res2a”)
reluLayer(“Name”, “res2a_relu”)];
lgraph = addLayers(lgraph, tempLayers);
tempLayers = [
convolution2dLayer([3,3], NF, “Name”, “res2b_branch2a”, “BiasLearnRateFactor”, 0, “Padding”,
[1 1 1 1])
batchNormalizationLayer(“Name”, “bn2b_branch2a”)
reluLayer(“Name”, “res2b_branch2a_relu”)
convolution2dLayer([3,3], NF, “Name”, “res2b_branch2b”, “BiasLearnRateFactor”, 0, “Padding”,
[1 1 1 1])
batchNormalizationLayer(“Name”, “bn2b_branch2b”)];
lgraph = addLayers(lgraph, tempLayers);
tempLayers = [additionLayer(2, “Name”, “res2b”)
reluLayer(“Name”, “res2b_relu”)];
lgraph = addLayers(lgraph, tempLayers);
tempLayers = [
convolution2dLayer([1,1], 2∗NF, “Name”, “res3a_branch1”, “BiasLearnRateFactor”, 0, “Stride”,
[2 2])
batchNormalizationLayer(“Name”, “bn3a_branch1”)];
```

```
lgraph = addLayers(lgraph, tempLayers);
tempLayers = [
convolution2dLayer([3,3], 2*NF, “Name”, “res3a_branch2a”, “BiasLearnRateFactor”, 0,
“Padding”, [1 1 1 1], “Stride”, [2 2])
batchNormalizationLayer(“Name”, “bn3a_branch2a”); reluLayer(“Name”, “res3a_branch2a_relu”)
convolution2dLayer([3,3],2*NF, “Name”, “res3a_branch2b”, “BiasLearnRateFactor”, 0,
“Padding”, [1 1 1 1])
batchNormalizationLayer(“Name”, “bn3a_branch2b”)];
lgraph = addLayers(lgraph, tempLayers);
tempLayers = [additionLayer(2, “Name”, “res3a”)
reluLayer(“Name”, “res3a_relu”)];
lgraph = addLayers(lgraph, tempLayers);
tempLayers = [
convolution2dLayer([3,3], 2*NF, “Name”, “res3b_branch2a”, “BiasLearnRateFactor”, 0,
“Padding”, [1 1 1 1])
batchNormalizationLayer(“Name”, “bn3b_branch2a”)
reluLayer(“Name”, “res3b_branch2a_relu”)
convolution2dLayer([3,3], 2*NF, “Name”, “res3b_branch2b”, “BiasLearnRateFactor”, 0,
“Padding”, [1 1 1 1])
batchNormalizationLayer(“Name”, “bn3b_branch2b”)];
lgraph = addLayers(lgraph, tempLayers);
tempLayers = [additionLayer(2, “Name”, “res3b”)
reluLayer(“Name”, “res3b_relu”)];
lgraph = addLayers(lgraph, tempLayers);
tempLayers = [
convolution2dLayer([1,1], 4*NF, “Name”, “res4a_branch1”, “BiasLearnRateFactor”, 0, “Stride”,
[2 2])
batchNormalizationLayer(“Name”, “bn4a_branch1”)];
lgraph = addLayers(lgraph, tempLayers);
tempLayers =
convolution2dLayer([3,3], 4*NF, “Name”, “res4a_branch2a”, “BiasLearnRateFactor”, 0,
“Padding”, [1 1 1 1], “Stride”, [2 2])
batchNormalizationLayer(“Name”, “bn4a_branch2a”)
reluLayer(“Name”, “res4a_branch2a_relu”)
convolution2dLayer([3,3], 4*NF, “Name”, “res4a_branch2b”, “BiasLearnRateFactor”, 0,
“Padding”, [1 1 1 1])
batchNormalizationLayer(“Name”, “bn4a_branch2b”)];
lgraph = addLayers(lgraph, tempLayers);
tempLayers = [additionLayer(2, “Name”, “res4a”)
reluLayer(“Name”, “res4a_relu”)];
lgraph = addLayers(lgraph, tempLayers);
tempLayers = [
convolution2dLayer([3,3], 4*NF, “Name”, “res4b_branch2a”, “BiasLearnRateFactor”, 0,
“Padding”, [1 1 1 1])
batchNormalizationLayer(“Name”, “bn4b_branch2a”)
reluLayer(“Name”, “res4b_branch2a_relu”)
convolution2dLayer([3,3], 4*NF, “Name”, “res4b_branch2b”, “BiasLearnRateFactor”, 0,
“Padding”, [1 1 1 1])
batchNormalizationLayer(“Name”, “bn4b_branch2b”)];
lgraph = addLayers(lgraph, tempLayers);
tempLayers = [ depthConcatenationLayer(2, “Name”, “depthcat”)
reluLayer(“Name”, “res5a_relu”)
averagePooling2dLayer([14,14], “Name”, “avgpool2d_2”, “Stride”, [14 14])
dropoutLayer(0.7, “Name”, “dropout”)
fullyConnectedLayer(26, “Name”, “fc”, “BiasLearnRateFactor”, 2, “WeightLearnRateFactor”, 2)
softmaxLayer(“Name”, “prob”); classificationLayer(“Name”, “classoutput”)];
lgraph = addLayers(lgraph, tempLayers);
lgraph = connectLayers(lgraph, “avgpool2d”, “res2a_branch2a”);
lgraph = connectLayers(lgraph, “avgpool2d”, “res2a/in2”);
lgraph = connectLayers(lgraph, “bn2a_branch2b”, “res2a/in1”);
lgraph = connectLayers(lgraph, “res2a_relu”, “res2b_branch2a”);
lgraph = connectLayers(lgraph, “res2a_relu”, “res2b/in2”);
lgraph = connectLayers(lgraph, “bn2b_branch2b”, “res2b/in1”);
lgraph = connectLayers(lgraph, “res2b_relu”, “res3a_branch1”);
lgraph = connectLayers(lgraph, “res2b_relu”, “res3a_branch2a”);
lgraph = connectLayers(lgraph, “bn3a_branch1”, “res3a/in2”);
lgraph = connectLayers(lgraph, “bn3a_branch2b”, “res3a/in1”);
lgraph = connectLayers(lgraph, “res3a_relu”, “res3b_branch2a”);
lgraph = connectLayers(lgraph, “res3a_relu”, “res3b/in2”);
lgraph = connectLayers(lgraph, “bn3b_branch2b”, “res3b/in1”);
lgraph = connectLayers(lgraph, “res3b_relu”, “res4a_branch1”);
lgraph = connectLayers(lgraph, “res3b_relu”, “res4a_branch2a”);
lgraph = connectLayers(lgraph, “bn4a_branch1”, “res4a/in2”);
lgraph = connectLayers(lgraph, “bn4a_branch2b”, “res4a/in1”);
lgraph = connectLayers(lgraph, “res4a_relu”, “res4b_branch2a”);
lgraph = connectLayers(lgraph, “res4a_relu”, “depthcat/in2”);
lgraph = connectLayers(lgraph, “bn4b_branch2b”, “depthcat/in1”);
% Clean Up Helper Variable
clear tempLayers;
```