Rethinking Recurrent Neural Networks and other Improvements for Image Classification

Nguyen Huu Phong  
University of Coimbra  
Centre of Informatics and Systems  
Portugal  
Email: phong@dei.uc.pt

Bernardete Ribeiro  
University of Coimbra  
Centre of Informatics and Systems  
Portugal  
Email: bribeiro@dei.uc.pt

Abstract—For a long history of Machine Learning which dates back to several decades, Recurrent Neural Networks (RNNs) have been mainly used for sequential data and time series or generally 1D information. Even in some rare researches on 2D images, the networks merely learn and generate data sequentially rather than for recognition of images. In this research, we propose to integrate RNN as an additional layer in designing image recognition’s models. Moreover, we develop End-to-End Ensemble Multi-models that are able to learn experts’ predictions from several models. Besides, we extend training strategy and softmax pruning which overall leads our designs to perform comparably to top models on several datasets. The source code of the methods provided in this article is available in https://github.com/leonlha/e2e-3m and http://nguyenhuuphong.me.

1. Introduction

In recent days, Image Recognition has been transformed into a new stage, thanks to the availability of high performance computing hardware, specially modern graphical processing units (GPUs) and large-scale datasets. Early design of Convolutional Neural Networks (ConvNets) in the nineties had only a few layers of depth, however, as more and more data with higher resolutions require much more computing power, the field has evolved to deeper/wider layers with more efficiency and accuracy [1], [2], [3], [4], [5], [6], [7]. Later developments involved balancing between number of networks’ depth, width and image resolution [3] or augmentation policies [9].

During the same time, Recurrent Neural Networks have been succeeded in various applications from Natural Language Processing [10], [11], Machine Translation [12], Speech Recognition [13], [14] to Weather Forecasting [15], Human Action Recognition [16], [17], [18], Drug Discovery [19] etc. However, in Image Recognition, RNN are merely used for generating sequences of image pixels [20], [21] instead of recognition of the whole image.

As architecture of RNN has been evolved and optimized via several revolutions, this could be interesting to research on weather these spectacular advances have a direct effect on image classification. Therefore, we take a distinct approach where RNN is integrated as an essential layer in designing image recognition models.

Besides, we propose an End-to-End (E2E) Ensemble Multiple Models that can learn expertise from various models. This results from our critical observation, when training models for specific datasets, oftentimes, we select the most accuracy models or ensemble some of them. However, we argue a merged predictions could provide a better solution than just one single model. Moreover, since ensembling essentially breaks the process (from getting input data until final predictions) into separated stages and may perform each step on different platforms, this causes a serious issue or even impossible to integrate the procedure into one place e.g. on Real-time systems [22], [23] or next platform - System on a Chip (SoC) [24], [25].

The rest of this article is organized as follows. Our main contributions are discussed in Section 2. RNN formulations including a typical RNN and more advanced RNNs will be focused in subsection 3.1. Besides, our essential idea of designing ConvNets models that can learn experiences from expert models are highlighted in subsection 3.2. Subsection 4.1 is where we initiate our design on iNaturalist’19 Dataset and in subsection 4.2 performance of various Image Recognition models integration with RNNs are thoroughly analysed. In subsection 4.3 we challenge our design on iCassava’19 Dataset. In subsection 4.4 and 4.5 we extend our discuss on learning rate strategy and softmax pruning technology. We conclude our research in Section 5.

2. Contributions

Our research differs from previous works in several ways. First, most of researches utilize RNNs for sequential data and time series. Even in rare cases, RNNs are just for generating sequences of image’s pixels. We propose to integrate RNNs as an essential layer of ConvNets. For evaluation, we perform our experiments on a virtual environment and a dedicated server using variety of latest fine-tuned models including InceptionV3, Xception, SEResNeXt101, EfficientNetB5, just to name a few.
Second, we present our vital idea for designing ConvNet in which the model is able to learn decisions from expert models. Typically, we just choose predictions from one single model or ensemble from some models. We improve our design after one challenge to the others.

Other main contributions of this work are a training strategy and an extension of softmax layer that allows our models to perform comparably to top models on several datasets.

Our source codes will be made available for other researchers to extend this work in any desirable directions. The programs are written in Jupyter Notebook environment using a web-based interface with a few extra libraries for facilitate reproduction process.

3. Methodologies

Our key idea is to integrate RNN as a layer in ConvNets models. Several RNNs are proposed and we present formulations for computations. The concept of training a model to learn the predictions from each individual model and its design is also discussed.

3.1. Recurrent Neural Networks

For the purpose of performance analysis and comparison, we choose a typical RNN along with more advanced ones i.e. Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) as well as Bi-direction RNN (BRNN). Formulations of respective RNNs are presented as follows.

Considering a standard RNN with a given input sequence $x_1, x_2, ..., x_T$, a hidden state of cell is updated at a time step $t$ as

$$h_t = \sigma(W_h h_{t-1} + W_x x_t + b) \quad (1)$$

where $W_h$, $W_x$ variables denote weight matrices and $b$ variable represents the bias, $\sigma$ is a sigmoid function which outputs values between 0 and 1.

The output of a cell, for easy of notation, is defined by

$$y_t = h_t \quad (2)$$

but also can be shown using softmax function in which $\hat{y}_t$ is the output and $y_t$ is the target instead.

$$\hat{y}_t = \text{softmax}(Wy_t + b_y) \quad (3)$$

A more sophisticated RNN or LSTM with introduction of forget gate can be expressed as in these equations:

$$f_t = \sigma(W_{fh} h_{t-1} + W_{fx} x_t + b_f) \quad (4)$$

$$i_t = \sigma(W_{ih} h_{t-1} + W_{ix} x_t + b_i) \quad (5)$$

$$c_t' = \tanh(W_{ch} h_{t-1} + W_{cx} x_t + b_c) \quad (6)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot c_t' \quad (7)$$

$$o_t = \sigma(W_{oh} h_{t-1} + W_{ox} x_t + b_o) \quad (8)$$

$$h_t = o_t \odot \tanh(c_t) \quad (9)$$

where operation $\odot$ represents the element-wise vector product; $f$, $i$, $o$ and $c$ are forget gate, input gate, output gate and cell state. Information is kept when the forget gate $f_t$ becomes 1 and is eliminated when $f_t$ is set to 0.

Since LSTM takes a huge resource for computing, another variation i.e. GRU is used for optimization by combining the input gate and forget gate into just one gate namely update gate. The mathematical formulas are expressed as follows:

$$r_t = \sigma(W_{rh} h_{t-1} + W_{rx} x_t + b_r) \quad (10)$$

$$z_t = \sigma(W_{zh} h_{t-1} + W_{zx} x_t + b_z) \quad (11)$$

$$h_t' = \tanh(W_{zh}(r_t \odot h_{t-1}) + W_{xh} x_t + b_z) \quad (12)$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot h_t' \quad (13)$$

Lastly, when a typical RNN essentially takes only previous information, Bi-direction RNNs integrate not only past but also future information

$$h_t = \sigma(W_{hx} x_t + W_{hh} h_{t-1} + b_h) \quad (14)$$

$$z_t = \sigma(W_{xh} x_t + W_{hh} h_{t+1} + b_z) \quad (15)$$

$$\hat{y}_t = \text{softmax}(Wy_h h_t + W_y z_t + b_y) \quad (16)$$

where $h_{t-1}$ and $h_{t+1}$ indicate cell hidden states at previous time step $t-1$ and future time step $t+1$.

For more detail of RNN, LSTM, GRU and BRNN please refer to articles [26], [27], [28], [29], [30], [26], [31] and [32], [33] respectively.

3.2. End-to-End Ensemble Multiple Models

Our main idea for this design is that, when we have several models trained on a certain dataset, typically we will choose the model that yields the best accuracy. However, we could build a model that can learn to combine expertise from all individual models. We illustrate this idea in Figure 1.

As shown in the upper part, each actor represents a trained single model. Every time, a sample is presented, each of these actors will predict how certain the sample falls into each category. These probabilities will be combined and utilized to train our model.

The bottom part of the Figure represents our ConvNets design for this idea. We essentially select three models for predictions instead of two. Since using only two models may result in the situation where one model dominates the other. Or in another word, we would have predictions mostly from only one model. Just, an additional model provides a balance between the two (we also limit our design to just three models because of resource limitation i.e. GPU memory). We name this model the E2E-3M. As the name suggests, E2E denotes the abbreviation of End-two-end learning process [34], [35], [36] where the model performs all phases from training until final prediction and 3M simply exhibits the combination of three models.

Each individual model (Net 1, Net 2 and Net 3) composes mainly a fine-tuned model where the last layer is removed and replaced by a more subtle layers e.g. a Global Pooling for reducing the size of the networks and RNN module (including a Reshape layer and an RNN layer). The
model also comprises a Gaussian Noise for preventing overfitting, a Fully Connected layer and their own Softmax layer. The outputs from the three models then are concatenated and utilized for training the following Neural Networks module which consist of a Fully Connected layer, a LeakyReLU layer, a Dropout layer and lastly a Softmax layer for classification.

Ensemble learning is one of key techniques in this design. Ensemble refers to aggregating weaker models (based learners) and construct a more efficient performance model (a stronger learner) [39]. The technique thrives better in Machine Learning than Deep Learning specially in Image Recognition since convolution requires a lot of computation power. Most recent and related researches focus on the use of a simple averaging or voting mechanism [40], [41], [42], [43], [44] and fewer investigate on integrating a trainable
Neural Networks [45], [46]. Our research differs from the others as we study the design on a much larger scale using a great number of up-to-date ConvNets.

Supposing that our ConvNets design have $n$ fine-tuned models or classifiers and $c$ classes for the dataset. The output of each classifier can be represented as a distribution vector:

$$\Delta_j = [\delta_{1j} \ \delta_{2j} \ \ldots \ \delta_{cj}] \quad (17)$$

where

$$1 \leq j \leq n$$

$$0 \leq \delta_{ij} \leq 1 \ \ \forall 1 \leq i \leq c$$

$$\sum_{i=1}^{c} \delta_{ij} = 1$$

After concatenation of $n$ classifiers, the distribution vector becomes:

$$\Delta = [\Delta_1 \ \Delta_2 \ \ldots \ \Delta_n] \quad (18)$$

For convenience of formulation, assuming that the Neural Networks has only one layer with number of neurons equals to the number of classes. As usual, the networks’ weights $\Theta$ are initiated randomly and the vector of distribution can be computed as:

$$\Delta' = \Theta \cdot \Delta = [\delta_{1}' \ \delta_{2}' \ \ldots \ \delta_{c}'] \quad (19)$$

where

$$1 \leq j \leq c$$

$$\delta_{j}' = \sum_{i=1}^{nc} \delta_{i}w_{ij}$$

Finally, the output after softmax activation:

$$\eta_{j}' = \frac{e^{\delta_{j}'}}{\sum_{i=1}^{c} e^{\delta_{i}'}} \quad (20)$$

4. Experiments

In this section, we will discuss experiments to evaluate our initial design without RNN, and analyse performance when RNNs are integrated. We also show our End-to-End Ensemble Multiple Models besides training strategy and extension of softmax layer. These experiments were performed on iNaturalist’19 [47] and iCassava’19 Challenges [48], Cifar-10 [49] and Fashion-MNIST [50] datasets.

4.1. Experiment 1

Deep Learning [51] and Convolutional Neural Networks (ConvNets) have shown notable successes in the field of Image Recognition. From LeNet [52], in several decades ago, until recently AlexNet [53], Inception [2], [3], [54], ResNet [4], SENet [7] and EfficientNet [8], these ConvNets leveraged automate classifications to exceed human’s performance in several applications. This efficiency is due to the highly availability of powerful computer hardware, specifically GPUs, and the Big Data.

In this subsection, we examine the significant of our design utilizing several ConvNets built based on leading architectures such as InceptionV3, ResNet50, InceptionResNetV2, Xception, MobileNetV1 and SEResNeXt101. We set InceptionV3 as a baseline model since the ConvNets has gained popularity among researchers in Deep Learning and can often be used as a standard testbed. Besides, InceptionV3 is known for employing sliding kernels e.g. $1 \times 1$, $3 \times 3$ or $5 \times 5$ in parallel which essentially reduces computation and increases accuracy. We also use a simplest version of Residual Networks i.e. ResNet50 where the brand (ResNet has several versions including ResNet50, ResNet101 and ResNet152 according to the numbers of depth layer) composes short circuits through each networks’ layer that reduces training time greatly. In addition, other ConvNets are explored for facilitating comparison and evaluation.

Our initial design is illustrated in Figure [2]. The center of this process implements one of models from above mentioned ConvNets. The architectures of these ConvNets differ from one to the other, but usually the top layers function as classifiers and can be replaced to adapt with different datasets. For example, Xception and ResNet50 comprise a Global Pooling and a Fully Connected in the top layers. On the other hand, VGG19 [55] constructs a Flatten and two Fully Connected layers (a Max Pooling in the original article but for some reasons Keras implements a Flatten layer).

In the design, we add a Global Pooling to decrease the output size of the networks (this is in line with most of ConvNets but contrasts to VGGs where exhausted Flatten layers are utilized). Importantly, we insert an RNN module for evaluation of our proposed approach. The module composes a Reshape layer and an RNN layer as described in subsection [3.1]. Moreover, a Gaussian noise layer is added to increase variation of samples for preventing overfitting. In the Fully Connected layer, the number of neurons e.g. 256, 512, 1024 or 2048 are varied which are driven mainly based on our experiments. The Softmax layer has a number of output according to the number of iNaturalist’19 categories.

All networks’ layers from the ConvNets are defrozen, so that we reuse only the models’ architectures and trained weights (these ConvNets are pre-trained on ImageNet dataset [56], [57]). Reusing trained weights offers several advantages since retraining or fine-tuning can increase accuracy. To distinguish these ConvNets with the original ones, we refer each model as a fine-tuned model.

Our experiments are performed on iNaturalist’19 dataset which is originally from iNaturalist Challenge conducted in Fine-Grained Visual Categorization 6th (FGVC6) workshop at CVPR 2019. In Computer Vision area, FGVC has attracted interest from researchers since around 2011 [58].
Figure 2. Single Model of E2E-3M. Fine-Tuned Model denotes a ConvNets (e.g. InceptionV3) with top layers are excluded and weights are retrained. The model is pre-loaded with ImageNet weights. The original images are rescaled to fit with a required input size of the Fine-tuned Model or for analysis of image resolution. Global Pooling layer reduces networks’ size. Reshaping converts data to a standard input for the RNN layer. Gaussian Noise layer expands variation of samples for preventing overfitting. Fully Connected layer aims to improve classification. Softmax layer is another Fully Connected layer which has a number of neurons as the number of dataset’s category and utilizes softmax activation.

[59], [60], though, researches on similar topics appeared long before [61], [62]. FGVC or subordinate categorization aims to classify visual objects at a more subtle detail level than basic level categories [63], for example, species of birds [58], dogs [60] and brands of cars [64], aircrafts [65]. In line with this development, iNaturalist dataset was created [47]. Besides, the dataset is comparable to ImageNet regarding size and category variation. The dataset used in this research - iNaturalist’19, focuses on more similar categories than previous versions and composed by 1010 species collected from approximately two hundred thousands of real plants and animals. Figure 3 shows random images from this dataset indicating species belonging to respective classes and sub-categories.

The dataset is split into training and test sets randomly with a ratio of 80/20. In addition, images are resized to several resolutions e.g. for InceptionV3 standard, rescaled images have a size of 299 × 299. The resolution is also increased to 401 × 401 or even 421 × 421. In Gaussian Noise layer, we set amount of noise at 0.1 and in Fully Connected layer, we choose 1024 neurons based on our experience since this is impractical to evaluate all layers with every settings.

We setup a Jupyter Notebook server which runs on a Linux Operation System (OS) using 4 GPUs (GeForce® GTX 1080 Ti Graphical Card) - 12 GB of RAM each. For coding, we use Keras with TensorFlow backend [66] as our platform. Keras was written in Python programming language and developed as an independent wrapper on top of other backends including TensorFlow. The project was acquired by Google Inc. and became one part of TensorFlow recently.

Figure 4 shows results for this experiment in which Top-1 Accuracy is plotted against Floating Point Operations Per Second (FLOPS). The size of each model or the total number of parameters are also displayed. The Top-1 Accuracy was obtained by submitting predictions to the Challenge website, obtaining Top-1 Error from private leader-board and subtracting the result by 1. During tournament, public leader-board is computed based on 51% of official test data. Aftermath, private leader-board is summarized with all data.

As we expected, a higher image resolution yields more accuracy but also uses more computing power for the same model (InceptionV3). As a side note, our benchmark has achieved an accuracy of 0.7097 with a small gap from the benchmark in Challenge website (0.7139). Since we are unaware of organizers’ ConvNets design, settings and working environment, we have no clue for the difference. Later on, we increase image’s resolutions from 299 × 299 to 401 × 401 and 421 × 421 and switch fine-tuned models. With Xception-421, our model has obtained the result of approximately 0.7347.

Since our server is shared, the training takes about one week each time. For this reason, the image size can be increased to only 421 × 421. Besides, results for SEResNeXt110-421, InceptionResNetV2-421 and ResNet50-299 are not obtained but the approximated accuracy are projected instead as we will be using these models in later experiments. Also, because adding an RNN module would increase the training time significantly, the module is not analyzed with this iNaturalist’19 dataset.

4.2. Experiment 2

As mentioned in the previous section, most of all researches regarding RNNs has focused on sequential data
or time series. Even with little attentions on images, the main purpose is to generate sequences of pixels rather than recognition of images. Our approach differs from the latter significantly where all image’s pixels are presented at once rather than at several time steps.

We will systematically evaluate our proposed design utilizing distinct Recurrent Neural Networks. These models include a typical RNN, an advanced GRU and a Bi-Directional RNN - BiLSTM, against a standard (STD) model where an RNN module is excluded. In addition, we select representative fine-tuned models namely InceptionV3, Xception, ResNet50, InceptionResNetV2, MobileNetV1, VGG19 and SEResNeXt101 for comparison and analysis.

In this experiment, we employ Fashion-MNIST dataset which is newly created by Zalando SE with the intention to serve as a direct replacement for MNIST dataset as a Machine Learning benchmark since MNIST has been achieved an almost perfect result of 100% accuracy. The Fashion-MNIST dataset has the same amount of data as MNIST including 50000 samples of training and 10000
samples of testing and also divided into 10 categories. Figure 5 visualizes how the dataset looks where each sample is a $28 \times 28$ grayscale image.

Our initial design (as discussed previously) is reused with a highlight note that RNN module has been incorporated. The number of unit for each RNN is set at 2048 (in BiLSTM, the number is 1024); the time step number is simply one for showing the whole image each time. In addition, because Fashion-MNIST dataset has the image size smaller than desired resolutions (e.g.: $244 \times 244$ or $299 \times 299$ for MobileNetV1 and InceptionV3), so all images are up-sampled.

We perform these experiments on Google Colab even though, our server runs faster. The primary reason is due to Jupyter Notebook occupies all GPUs for the first login section. In other words, only one program executes with full capability. This opposes to the Colab where multiple environments are able to run in parallel. The second reason is the virtual environment allows rapid developments i.e. installation of additional libraries and run programs instantly. All experiments are set for 12 hours duration since some take less time before overfitting but few others take more than the maximum time allotted. We repeat each experiment 3 times. Moreover, in this research, we often submit results to challenge websites, we record only the highest accuracy rather than using other measurements. The accuracy measurement is defined as follows:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total numbers of predictions made}}$$  \hspace{1cm} (21)

Table 1 shows comparisons of models using different RNN modules on InceptionV3 and MobileNetV1. The former is often chosen as a baseline model whereas the latter is the lightest model in terms of parameters and computation. The results are shown on Figure 6 for convenient observation. As we can see, models with the additional RNN modules can surprisingly achieve higher accuracy than STD models.

We extend this experiment and include more models including Xception, ResNet50, InceptionResNetV2, VGG19 and SEResNeXt101. Table 2 shows comparisons of models

---

1. https://colab.research.google.com/. Google Colab was started as an internal project built based on Jupyter Notebook and was provided to public in 2018. At the time this article is written, the virtual environment supports a single 12GB NVIDIA Tesla K80 GPU.

TABLE 1. ACCURACY COMPARISON OF MODELS USING DISTINCT RECURRENT NEURAL NETWORKS ON INCEPTIONV3 AND MOBILENETV1

| Fine-tuned Models   | STD  | RNN  | GRU  | BiLSTM |
|---------------------|------|------|------|--------|
| InceptionV3         | 0.9434 | 0.9453 | 0.9430 | 0.9445 |
| MobileNetV1         | 0.9404 | 0.9438 | 0.9439 | 0.9410 |

Figure 4. iNaturalist'19 Accuracy of Models. Benchmark denotes the result for InceptionV3 with the default setting (image resolution $299 \times 299$). In addition, the sizes of input image in other models are indicated along with respective names. For example, Xception-421 indicates the input image for the model which has been rescaled to $421 \times 421$. 

TABLE 2.
integrated with BiLSTM module versus standard models. In most of all models, BiLSTMs significantly outperform STDs, except InceptionResNetV2. Please note that, VGG19 took a tremendous training time, eventually, the model has not reached the accuracy as other models after 12 hours of training. In the same manner, we could not obtain a result for VGG19-BiLSTM as well as results for SEResNeXt101. Figure 7 shows only the results for InceptionV3, Xception, ResNet50, InceptionResNetV2 and MobileNetV1.

4.3. Experiment 3

Often times, when training a model, we randomly split a dataset into train and test sets with a desired ratio. Then we repeat our evaluation several times and finally obtaining results from one of measurement methods e.g. mean or standard deviation accuracy. However, in competitions like
Figure 6. Accuracy Comparison of Models Integrated with Recurrent Neural Networks vs Standard Model. Integration models significantly outperform STD models.

TABLE 2. COMPARISON OF BiLSTM AND STD ON DIFFERENT FINE-TUNED MODELS INCLUDING InceptionV3, Xception, ResNet50, InceptionResNetV2, MobileNetV1, VGG19 AND SEResNeXt101.

| Fine-tuned Models | InceptionV3 | Xception | ResNet50 | InceptionResNetV2 |
|-------------------|-------------|-----------|----------|-------------------|
| STD               | 0.9434      | 0.9374    | 0.9375   | 0.9364            |
| BiLSTM            | 0.9445      | 0.9385    | 0.9377   | 0.9360            |
| MobileNetV1       | 0.9404      | 0.9075    | N/A      | N/A               |
| VGG19             | 0.9410      | N/A       | N/A      |                   |
| SEResNeXt101      |             |           |          |                   |

the ones on Kaggle, a test set is completely separated from a train set. If we naively divide the train set into another train and a valid set, we will face a loophole that all samples of the original train set may not be used for training since a portion of dataset is always needed for validating. To solve this problem, we apply k-folds validation procedure on the train set by dividing the dataset into k subsets and one subset is taken out for validation. We expand this process and make predictions also on the official test set. This way allows our models (one model for each set) to learn from all images in the official train set.

We also attempt to increase robustness performance of models via the use of data augmentation. The technique aims to transform the original training dataset, create an expanded dataset whose true labels are knew [67], [68], [69]. Importantly, this teaches the model invariant or irrelevant of input variations [70]. For example, flipping an image of a car horizontally does not change the corresponding category. We apply an augmentation approach from [53] on test set in which a sample is cropped multiple times and predictions are made for each instance. The procedure lately becomes a standard practice in image recognition and referred as Test Time Augmentation. In our research, we crop images at random locations instead of at only four corners and the center. Besides, the most prevalent augmentation techniques for geometric and texture transformations such as rotation, width/height shifts, shear and zoom, horizontal/vertical flips and channel shift are utilized.

In addition, we apply state-of-the-art ensemble learning since the technique is generally more accuracy than prediction from a single model. We use a simple averaging approach for aggregation of all models a.k.a AVG-3M. Also, this is important to ensure a variety of fine-tuned models to increase the diversity of classifiers as combining multiple redundant classifiers would be meaningless. We finally choose SEResNeXt101, Xception and InceptionResNetV2 fine-tuned models as these ConvNets yield higher results than others. Please note the RNN module (including Reshape and BiLSTM layers) is excluded.

One more crucial technique is our training strategy that helps to reduce lost and increase accuracy by searching for a better global minimum (more details will be analyzed in next subsection).

We evaluate our approach using iCassava Challenge’s dataset which was also organized by FGVC6 workshop at CVPR19. In iCassava dataset, leaf images of cassava plants are divided into 4 categories of diseases namely
Cassava Mosaic Disease, Cassava Green Mite disease, Cassava Bacterial Blight, Cassava Brown Streak Disease and 1 category of healthy plant with total 9,436 images labeled. The challenge was organized on Kaggle website \(^2\) run from 26th April to 2nd June 2019 and attracted nearly 100 teams around the world. Proposed models are evaluated on 3,774 official test data and the results were submitted to the website. Public leader-board is summarized based on 40% of test data whereas private leader-board is computed with all test data. Figure 8 shows random samples of this dataset.

The iCassava official train set is splitted into 5 subsets for k-fold cross validation in which one subset is held out for testing and the others for training in turn. We perform these experiments on a server using GeForce\textsuperscript{\textregistered} GTX 1080 Ti graphical card with 4 GPUs - 12 GB of RAM each.

When evaluating on the test set, images are up-sampled to a higher resolution (540 \times 540) and randomly cropped to the input size (501 \times 501). We vary numbers of crops including 1, 3, 5, 7 and 9. We finally select just 3 crops as this choice yields higher accuracy than other cropping choices in most of subsets. We obtain the results using two methods, one without cropping and the other with 3 crops and select the one with higher accuracy for each set. Then results for all sets are averaged to provide a result for the whole dataset. To this step, the result has achieved 0.928.

Furthermore, we apply AVG-3M design on Set number 3, 4 and 5; and average each of these results with all other sets. Table 3 shows results for this experiment. As we can glean from the table, using our AVG-3M for these sets improves accuracy. The combination of AVG-3M from Set 4 with remaining sets yields the highest result 0.9368. This is critical to notice that, even though the public leader-board result for all sets with AVG-3M of Set 3 yields a higher result than that of Set 4, but eventually, the latter has achieved a better result. Therefore, choosing submissions for final evaluation is efficient and essential. Figure 9 shows our results for comparison with other top-10 teams.

4.4. Experiment 4

In this sub-section, we deal with overall improvement of our model using learning rate strategy. We apply Adam optimizer \(^7\) as one of advanced optimizers in Deep Learning area. The computations are as follows:
Figure 8. Random Samples from iCassava Dataset. CMD, CGM, CBB and CBSD denotes Cassava Mosaic Disease, Cassava Green Mite disease, Cassava Bacterial Blight, Cassava Brown Streak Disease, respectively.

\[ w_t = w_{t-1} - \eta_t \cdot \frac{m_t}{\sqrt{v_t + \epsilon}} \]  
\[ \eta_t = \eta \cdot \sqrt{1 - \beta_2^t} \]  
\[ m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \]  
\[ v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \]

where \( w \) and \( \eta \) is weight and learning rate of the Neural Networks; \( m \), \( v \) and \( g \) are moving averages and gradient of current mini-batch; betas \( (\beta_1, \beta_2) \) and epsilon \( \epsilon \) are set at 0.9, 0.999 and \( 10^{-8} \), respectively.

We use Keras to implement the models and, in the platform, a formula for computing the learning rate with decay is:

\[ \eta = \eta \cdot \frac{1}{1 + \text{decay} \cdot \text{iterations}} \]
Choosing learning rate is essential and critical as training time often is reduced greatly with a correct learning rate. However, selecting an appropriate learning rate is difficult as if the step is very big then the global minimum might be ignored. On the other hand, a very small step could cause an extremely long training time. In our experiments, we started with a moderate learning rate and train until accuracy stops to improve, then we reduce the learning rate, reload ConvNets’ weights with the highest accuracy and repeat this process for the second and third time. Figure 10 illustrates our proposed training direction.

We evaluate these experiments on Cifar-10 and Fashion-MNIST datasets. Initially, the learning rate is set at $1 \times 10^{-4}$ and then changed to $1 \times 10^{-5}$ and $1 \times 10^{-6}$ sequentially. The decay is derived by dividing the learning rate by the number of epochs.

Figure 11 shows the performances of top-3 models on Fashion-MNIST using SEResNeXt101 STD, Xception LSTM and InceptionResNetV2 STD and excludes other ConvNets discussed in previous Sections. The three variations of RNN also analyzed against STD models and only the model with highest accuracy is shown on the figure. As we can see, these accuracy are effectively increased after the transition. SEResNeXt101 has achieved a result of 0.9541 in STD setting.

This result is further extended to 0.9585 using E2E-3M model as the design is shown in Figure 3.2 and steps are detailed in Algorithm 1 with settings as follows. The Fully Connected layer (after the Concatenate layer) has 4096 neurons where as the LeakyReLU has a slope of 0.2 and Dropout is set at 0.5. Please note that for reloading weights, sometimes we need to convert these weights to Pickle format rather than Keras’ standard HDF5 since networks’ weights are too large.

In the same manner, we perform experiments on Cifar-10 dataset and results are shown on Figure 12. Please note that, EfficientNetB5 is also added as the ConvNets is one of the latest models in the field. Using EfficientNetB5, we achieve a result of 0.9788 on STD setting.

### 4.5. Experiment 5

In this subsection, our setup include a variation of softmax layer where only outputs of the most active neurons are used for prediction. We observe, in multi-categories prediction, often, a few or even only one confidence on a category is large enough to be meaningful while others are very small i.e. nearly zero percent of confidence. For this reason, we propose to eliminate these meaningless confidences by zeroing them before using for ensembling. We compare this approach (namely EXT-Softmax) with a typical method where multiple predictions are averaged for a final prediction (AVG-Softmax). The essential steps are illustrated in Algorithm 2.

Results on Fashion-MNIST are 0.9592 and 0.9591 without improvement on the proposed approach. However on Cifar-10, the accuracy is higher from 0.9833 to 0.9836. Table 4 and 5 show latest achievements on these datasets. Please note that the results on Fashion-MNIST were voluntarily submitted and were not officially verified. Though,
we went through each profile and selected only results that are supported by publications. Importantly, the dataset was changed recently because of duplication, therefore, our result could be higher if the previous version was used.

5. Conclusions

In this research, we present our vital ideas for improvements of ConvNets from integration of RNN as an essential layer in ConvNets, the design of End-to-End Ensemble Multiple Models for learning expertise from each individual ConvNets to the training strategy and the concept for extension of Softmax layer.

First, we propose the integration of RNN into ConvNets even though RNNs are mainly optimized for 1D sequential data rather than 2D images. Our results on Fashion-MNIST shows that ConvNets with RNN, GRU and BiLSTM modules can outperform standard ConvNets using variety of fine-tuned Models including InceptionV3, Xception, ResNet50, InceptionResNetV2 and MobileNetV1 on a limited virtual environment. Similar results can be obtained on a dedicated server for SEResNeXt101 and EfficientNetB5 using Cifar-10 and Fashion-MNIST datasets. Though, adding RNN modules requires more computing power, this can be a trade of between accuracy and running time.

Second, we design the E2E-3M ConvNets that is able to learn predictions from several models. The design is built along with this research by aggregating and analysing each module step by step. We initially test the model on iNaturalist’19 dataset using only one single model of E2E-3M. Various models and image resolutions are evaluated and compared with the Inception Benchmark model from the challenge. The model then is added with RNNs and the performance is analyzed on Fashion-MNIST dataset. This is also interesting when we joined iCassava’19 challenge where our model is further extended. In addition, our E2E-3M model outperforms a standard single model by a large margin. Using End-two-end design also allows the model to run instantly on Real-time or System-on-a-Chip platforms.

Finally, we propose the training strategy and the pruning for softmax layer which yields comparable accuracy on Cifar-10 and Fashion-MNIST.
In the future, we plan to extend our models to more variety of settings. For example, we would evaluate BiRNN and BiGRU modules. In addition, we eager to analyze optimizers e.g. AdaDelta, AdaGrad, RMSprop or even a simple version SGD.

Acknowledgments

We thank University of Coimbra, CISUC - Center of Informatics and Systems for financial support. Also the reviewers are gratefully acknowledged for their insightful comments.

References

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

[2] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Re-thinking the inception architecture for computer vision,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 2818–2826.

[3] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, “Inception-v4, inception-resnet and the impact of residual connections on learning,” in Thirty-first AAAI conference on artificial intelligence, 2017.

[4] 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.

[5] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, “Aggregated residual transformations for deep neural networks,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 1492–1500.

[6] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 1251–1258.

[7] J. Hu, L. Shen, and G. Sun, “Squeeze-and-excitation networks,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 7132–7141.

[8] M. Tan and Q. V. Le, “EfficientNet: Rethinking model scaling for convolutional neural networks,” arXiv preprint arXiv:1905.11946, 2019.

[9] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, “Autoaugment: Learning augmentation strategies from data,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2019, pp. 113–123.

[10] P. Liu, X. Qiu, and X. Huang, “Recurrent neural network for text classification with multi-task learning,” in Proceedings of International Joint Conference on Artificial Intelligence, 2016.

[11] W. Yin, K. Kann, M. Yu, and H. Schütze, “Comparative study of cnn and rnn for natural language processing,” arXiv preprint arXiv:1702.01923, 2017.

[12] S. Chopra, M. Auli, and A. M. Rush, “Abstractive sentence summarization with attentive recurrent neural networks,” in Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2016, pp. 93–98.
Figure 12. Accuracy of Models on Cifar-10 using Training Strategy. The models are trained with learning rate starts at $1 \times 10^{-4}$ during 40 epochs. RNN modules including STD, RNN, LSTM and GRU are compared. The best module is selected for each model e.g. STD in case of EfficientNetB5. The models are reloaded with the highest check points and continue to train again for second and third time for 15 epochs each with learning rates of $1 \times 10^{-5}$ and $1 \times 10^{-6}$, respectively.

[13] C.-C. Chiu, T. N. Sainath, Y. Wu, R. Prabhavalkar, P. Nguyen, Z. Chen, A. Kannan, R. J. Weiss, K. Rao, E. Gonina et al., “State-of-the-art speech recognition with sequence-to-sequence models,” in 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2018, pp. 4774–4778.

[14] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, “Listen, attend and spell: A neural network for large vocabulary conversational speech recognition,” in 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2016, pp. 4960–4964.

[15] X. Qing and Y. Niu, “Hourly day-ahead solar irradiance prediction using weather forecasts by lstm,” Energy, vol. 148, pp. 461–468, 2018.

[16] P. Zhang, C. Lan, J. Xing, W. Zeng, J. Xu, and N. Zheng, “View adaptive recurrent neural networks for high performance human action recognition from skeleton data,” in Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 2117–2126.

[17] S. Song, C. Lan, J. Xing, W. Zeng, and J. Liu, “An end-to-end spatio-temporal attention model for human action recognition from skeleton data,” in Thirty-first AAAI conference on artificial intelligence, 2017.

[18] M. Liu, H. Liu, and C. Chen, “Enhanced skeleton visualization for view invariant human action recognition,” Pattern Recognition, vol. 68, pp. 346–362, 2017.

[19] M. H. Segler, T. Kogej, C. Tyrchan, and M. P. Waller, “Generating focused molecule libraries for drug discovery with recurrent neural networks,” ACS central science, vol. 4, no. 1, pp. 120–131, 2018.

[20] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra, “Draw: A recurrent neural network for image generation,” arXiv preprint arXiv:1502.04623, 2015.

[21] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu, “Pixel recurrent neural networks,” arXiv preprint arXiv:1601.06759, 2016.

[22] X. Wang, W. Zhang, X. Wu, L. Xiao, Y. Qian, and Z. Fang, “Real-time vehicle type classification with deep convolutional neural networks,” Journal of Real-Time Image Processing, vol. 16, no. 1, pp. 5–14, 2019.

[23] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang, “Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 1874–1883.

[24] D. Giri, K.-L. Chiu, G. Di Guglielmo, P. Mantovani, and L. P. Carloni, “Esp4ml: Platform-based design of systems-on-chip for embedded machine learning,” arXiv preprint arXiv:2004.03640, 2020.

[25] Y. Zhu, M. Mattina, and P. Whatmough, “Mobile machine learning hardware at arm: a systems-on-chip (soc) perspective,” arXiv preprint arXiv:1801.06274, 2018.

[26] L. Jing, C. Gulcehre, J. Peurifoy, Y. Shen, M. Tegmark, M. Soljacic, and Y. Bengio, “Gated orthogonal recurrent units: On learning to forget,” Neural computation, vol. 31, no. 4, pp. 765–783, 2019.

[27] H. Salehinejad, S. Sankar, J. Barfett, E. Colak, and S. Valaee, “Recent advances in recurrent neural networks,” arXiv preprint arXiv:1801.01078, 2017.

[28] Y. Yu, X. Si, C. Hu, and J. Zhang, “A review of recurrent neural networks: Lstm cells and network architectures,” Neural computation, vol. 31, no. 7, pp. 1235–1270, 2019.

[29] A. Graves, “Generating sequences with recurrent neural networks,” arXiv preprint arXiv:1308.0850, 2013.

[30] F. A. Gers, J. Schmidhuber, and F. Cummins, “Learning to forget: Continual prediction with lstm,” 1999.
**TABLE 3.** ICASSAVA Result. The dataset is divided into 5 subsets in which 4 subsets are combined as a train set and the other subset is used as a valid set. Each of these combinations (including the train and valid set) comprises a dataset namely from Set 1 to Set 5. Valid denotes results obtained using valid data. Private and Public represent results obtained from corresponding categories on the challenge website. Test - 0 Crop means the test results without cropping whereas Test - 3 Crop implies the test results using 3 crops method. Best results for each subset are summarized on the table at bottom left including Private and Public results. The table on bottom right represents the results obtained using AVG-3M for each individual set and when combines each result with all the other.

| Models                  | Set 1        | Set 2        | Set 3        | Set 4        | Set 5        |
|-------------------------|--------------|--------------|--------------|--------------|--------------|
| **SEResNeXt 101**       | Valid        | Set 1        | Set 2        | Set 3        | Set 4        | Set 5        |
|                         | 0.9043       | 0.9105       | 0.8963       | 0.9025       | 0.9061       |
|                         | Private      | Public       | Private      | Public       | Private      | Public       |
|                         | 0.9019       | 0.8874       | 0.9129       | 0.8947       | 0.9041       | 0.8874       |
| Test - 0 Crop           | 0.9023       | 0.8887       | 0.9143       | 0.9026       | 0.9015       | 0.888        |
| Test - 3 Crop           |              |              |              |              |              |              |
|                         |              |              |              |              |              |              |
| **Xception**            | Valid        | Test - 3 Crop| Private      | Public       | Private      | Public       |
|                         |              |              | 0.9025       | 0.914        | 0.9158       |
|                         | Private      | Public       | Private      | Public       | Private      | Public       |
|                         |              |              |              |              |              |              |
| Test - 0 Crop           |              |              |              |              |              |              |
| Test - 3 Crop           |              |              |              |              |              |              |
| **InceptionResNet V2**  | Valid        | Test - 3 Crop| Private      | Public       | Private      | Public       |
|                         |              |              | 0.9043       | 0.9078       | 0.9114       |
|                         | Private      | Public       | Private      | Public       | Private      | Public       |
|                         |              |              |              |              |              |              |
| Test - 0 Crop           |              |              |              |              |              |              |
| Test - 3 Crop           |              |              |              |              |              |              |

**Algorithm 1 E2E-3M**

1. **Input:** A training set with c categories D.
2. Select top 3 models.
3. Step 2: Train m fine-tuned networks.
4. Step 3: Select top-3 models.
5. Step 4: Reload weights of the three models.
6. Step 5: Construct a new dataset of predictions.
7. Step 6: Learn. (Algorithm 1)
8. Step 7: Apply LeakyReLU activation (slope = k).
9. Step 8: Regularize using Dropout (rate = p).
10. Step 9: Learn Final Fully Connected Neural Networks.
11. Step 10: Apply Softmax.
12. Step 11: Return A.

where:

- \( z_i \) is a gating variable 0-1 follows Bernoulli distribution with \( \beta_i = \{ \mu_1, \gamma \} \).
- \( \gamma = \{ \Delta_1 \} \).
- \( \Delta_1 = \sum_{i=0}^{1} \delta_{w_i} \).
- \( \delta_{w_i} \leq \lambda \).
- \( \delta_{w_i} \geq \lambda \).
- \( \Delta_1 = \{ 0 \} \).
- \( M_i = \{ \mu_1, \gamma \} \).
- \( \mu_1 = \{ \Delta_1 \} \).
- \( \gamma = \{ \Delta_1 \} \).
- \( \Delta_1 = \sum_{i=0}^{1} \delta_{w_i} \).

**Algorithm 2 Pruning Output: Pruned A**

1. **Function PruneNet(A):**
2. \( A = \text{PruneNet}(A) \).
3. \( N = \text{length}(A) \).
4. \( \text{for} \ i = 1 \text{ to } N \text{ do} \)
5. \( \text{if } A_i = \text{indexMax} \text{ then} \)
6. \( \text{indexMax} = \text{indexMax}(A) \).
7. \( \text{end if} \)
8. \( \text{end for} \)
9. \( \text{return } A \).

where:

- \( \mu_1 \) is the pruned output.
- \( \gamma = \{ \Delta_1 \} \).
- \( \Delta_1 = \sum_{i=0}^{1} \delta_{w_i} \).
- \( \delta_{w_i} \leq \lambda \).
- \( \delta_{w_i} \geq \lambda \).
- \( \Delta_1 = \{ 0 \} \).
- \( M_i = \{ \mu_1, \gamma \} \).
- \( \mu_1 = \{ \Delta_1 \} \).
- \( \gamma = \{ \Delta_1 \} \).
- \( \Delta_1 = \sum_{i=0}^{1} \delta_{w_i} \).

**Algorithm 3 SoftMax:**

1. **Step 1:** Learn Level-1 classifiers.
2. **Step 2:** Train m fine-tuned models.
3. **Step 3:** Select top-3 models.
4. **Step 4:** Reload weights of the three models.
5. **Step 5:** Construct a new dataset of predictions.
6. **Step 6:** Learn. (Algorithm 1)
7. **Step 7:** Apply LeakyReLU activation (slope = k).
8. **Step 8:** Regularize using Dropout (rate = p).
9. **Step 9:** Learn Final Fully Connected Neural Networks.
10. **Step 10:** Apply Softmax.

where:

- \( z_i \) is a gating variable 0-1 follows Bernoulli distribution with \( \beta_i = \{ \mu_1, \gamma \} \).
- \( \gamma = \{ \Delta_1 \} \).
- \( \Delta_1 = \sum_{i=0}^{1} \delta_{w_i} \).
- \( \delta_{w_i} \leq \lambda \).
- \( \delta_{w_i} \geq \lambda \).
- \( \Delta_1 = \{ 0 \} \).
- \( M_i = \{ \mu_1, \gamma \} \).
- \( \mu_1 = \{ \Delta_1 \} \).
- \( \gamma = \{ \Delta_1 \} \).
- \( \Delta_1 = \sum_{i=0}^{1} \delta_{w_i} \).
TABLE 4. LIST OF LATEST ACHIEVEMENTS ON FASHION-MNIST. THE RESULTS WERE SUBMITTED TO OFFICIAL WEBSITE OF FASHION-MNIST DATASET. CLASSIFIER INDICATES THE MAIN METHOD THAT WAS USED TO ACHIEVE THE RESULT.

| Classifier                  | Accuracy | Submitter       |
|-----------------------------|----------|-----------------|
| WRN-28-10 + Random Erasing  | 0.965    | @zhunzhong07    |
| WRN-28-10                   | 0.959    | @zhunzhong07    |
| Dual path network with WRN-28-10 | 0.957 | @Quesqueg       |
| DENSER                      | 0.953    | @fillassuncao   |
| MobileNet                   | 0.950    | @Bojone         |
| CNN with optional shortcuts | 0.947    | @kennivich      |
| Google AutoML               | 0.939    | @Sebastian Heinz|
| Capsule Network             | 0.936    | @XifengGuo      |
| VGG16                       | 0.935    | @QuantumLiu     |
| LeNet                       | 0.934    | @cmasch         |
| AVG-Softmax                 | 0.9592   | N/A             |
| EXT-Softmax                 | 0.9591   | N/A             |
| E2E-3M                      | 0.9585   | N/A             |
| SeResNeXt101-STD            | 0.9541   | N/A             |

TABLE 5. LIST OF RECENT ACHIEVEMENTS ON CIFAR-10 ALONG WITH RESULTS FROM OUR MODELS. THE PROPOSED APPROACH PERFORMS COMPARABLY TO THE TOP MODELS.

| Authors                     | Accuracy (%) |
|-----------------------------|--------------|
| Yanping Huang et al.        | 99.00        |
| Ekin D. Cubuk et al.         | 98.52        |
| Niv Nayman et al.            | 98.40        |
| SE-ResNeXt101-GRU            | 98.36        |
| EfficientNetB5-STD           | 98.33        |
| Yoshio Yamada et al.         | 97.69        |
| Terrance DeVries et al.      | 97.44        |
| SE-ResNeXt101-GRU            | 97.31        |
| Zhan Zhong et al.            | 96.92        |
| Senwei Liang et al.          | 96.55        |
| Gao Huang et al.             | 96.54        |
| Benjamin Graham              | 96.53        |
| Ke Zhang et al.              | 96.23        |

[31] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using rnn encoder-decoder for statistical machine translation,” arXiv preprint arXiv:1406.1078, 2014.

[32] Z. C. Lipton, J. Berkowitz, and C. Elkan, “A critical review of recurrent neural networks for sequence learning,” arXiv preprint arXiv:1506.00019, 2015.

[33] M. Schuster and K. K. Paliwal, “Bidirectional recurrent neural networks,” IEEE transactions on Signal Processing, vol. 45, no. 11, pp. 2673–2681, 1997.

[34] Y. He, T. N. Sainath, R. Prabhavalkar, I. McGraw, R. Alvarez, D. Zhao, D. Rybach, A. Kannan, Y. Wu, R. Pang et al., “Streaming end-to-end speech recognition for mobile devices,” in ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2019, pp. 6381–6385.

[35] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, and U. Muller, “Explaining how a deep neural network trained with end-to-end learning steers a car,” arXiv preprint arXiv:1704.07911, 2017.

[36] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang et al., “End to end learning for self-driving cars,” arXiv preprint arXiv:1604.07316, 2016.

[37] A. L. Maas, A. Y. Hannun, and A. Y. Ng, “Rectifier nonlinearities improve neural network acoustic models,” in Proc. icml, vol. 30, no. 1, 2013, p. 3.

[38] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” The journal of machine learning research, vol. 15, no. 1, pp. 1929–1958, 2014.

[39] O. Sagi and L. Rokach, “Ensemble learning: A survey,” Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, vol. 8, no. 4, p. e1249, 2018.

[40] F. Perez, S. Avila, and E. Valle, “Solo or ensemble? choosing a cnn architecture for melanoma classification,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2019, pp. 0–0.

[41] G. Antipov, S.-A. Berrani, and J.-L. Dugelay, “Minimalistic cnn-based ensemble model for gender prediction from face images,” Pattern recognition letters, vol. 70, pp. 59–65, 2016.

[42] M. Duan, K. Li, and K. Li, “An ensemble cnn2elm for age estimation,” IEEE Transactions on Information Forensics and Security, vol. 13, no. 3, pp. 758–772, 2017.

[43] D. Vasan, M. Alazah, S. Wassan, B. Safaei, and Q. Zheng, “Image-based malware classification using ensemble of cnns architectures (invecc),” Computers & Security, p. 101748, 2020.

[44] R. Wang, W. Li, and L. Zhang, “Blur image identification with ensemble convolutional neural networks,” Signal Processing, vol. 155, pp. 73–82, 2019.

[45] M. Amin-Naji, A. Aghagolzadeh, and M. Ezoji, “Ensemble of cnn for multi-focus image fusion,” Information fusion, vol. 51, pp. 201–214, 2019.

[46] Y. Fan, J. C. Lam, and V. O. Li, “Multi-region ensemble convolutional neural network for facial expression recognition,” in International Conference on Artificial Neural Networks. Springer, 2018, pp. 84–94.

[47] G. Van Horn, O. Mac Aodha, Y. Song, Y. Cui, C. Sun, A. Shepard, H. Adam, P. Perona, and S. Belongie, “The inaturalist species classification and detection dataset,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 8769–8778.

[48] E. Mwebaze, T. Gebru, A. Frome, S. Nsamba, and J. Tusubira, “icassava 2019 fine-grained visual categorization challenge,” arXiv preprint arXiv:1908.02900, 2019.

[49] A. Krizhevsky, G. Hinton et al., “Learning multiple layers of features from tiny images,” 2009.

[50] “Fashion-mnist,” https://github.com/zalandoresearch/fashion-mnist, accessed: 2020-04-20.

[51] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” nature, vol. 521, no. 7553, pp. 436–444, 2015.

[52] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Backpropagation applied to handwritten zip code recognition,” Neural computation, vol. 1, no. 4, pp. 541–551, 1989.

[53] 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.

[54] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 1–9.

[55] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” in European conference on computer vision. Springer, 2014, pp. 818–833.
[57] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein et al., “Imagenet large scale visual recognition challenge,” International journal of computer vision, vol. 115, no. 3, pp. 211–252, 2015.

[58] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, “The caltech-ucsd birds-200-2011 dataset,” 2011.

[59] B. Yao, A. Khosla, and L. Fei-Fei, “Combining randomization and discrimination for fine-grained image categorization,” in CVPR 2011. IEEE, 2011, pp. 1577–1584.

[60] A. Khosla, N. Jayadevaprakash, B. Yao, and F.-F. Li, “Novel dataset for fine-grained image categorization: Stanford dogs,” in Proc. CVPR Workshop on Fine-Grained Visual Categorization (FGVC), vol. 2, no. 1, 2011.

[61] C. A. Collin and P. A. McMullen, “Subordinate-level categorization relies on high spatial frequencies to a greater degree than basic-level categorization,” Perception & psychophysics, vol. 67, no. 2, pp. 354–364, 2005.

[62] C. R. Gillebert, H. P. Op de Beeck, S. Panis, and J. Wagemans, “Subordinate categorization enhances the neural selectivity in human object-selective cortex for fine shape differences,” Journal of Cognitive Neuroscience, vol. 21, no. 6, pp. 1054–1064, 2009.

[63] A. B. Hillel and D. Weinshall, “Subordinate class recognition using relational object models,” in Advances in Neural Information Processing Systems, 2007, pp. 73–80.

[64] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, “3d object representations for fine-grained categorization,” in 4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13), Sydney, Australia, 2013.

[65] S. Maji, J. Kannala, E. Rahtu, M. Blaschko, and A. Vedaldi, “Fine-grained visual classification of aircraft,” Tech. Rep., 2013.

[66] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin et al., “Tensorflow: Large-scale machine learning on heterogeneous distributed systems,” arXiv preprint arXiv:1603.04467, 2016.

[67] P. T. Jackson, A. A. Abarghouei, S. Bonner, T. P. Breckon, and B. Obara, “Style augmentation: data augmentation via style randomization.” in CVPR Workshops, 2019, pp. 83–92.

[68] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang, “Random erasing data augmentation.” in AAAI, 2020, pp. 13 001–13 008.

[69] R. G. Lopes, D. Yin, B. Poole, J. Gilmer, and E. D. Cubuk, “Improving robustness without sacrificing accuracy with patch gaussian augmentation,” arXiv preprint arXiv:1906.02611, 2019.

[70] Y. LeCun, “Learning invariant feature hierarchies,” in European conference on computer vision. Springer, 2012, pp. 496–505.

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

[72] S. Liang, Y. Khoo, and H. Yang, “Drop-activation: Implicit parameter reduction and harmonic regularization,” arXiv preprint arXiv:1811.05850, 2018.

[73] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 4700–4708.

[74] B. Graham, “Fractional max-pooling,” arXiv preprint arXiv:1412.6071, 2014.

[75] K. Zhang, M. Sun, T. X. Han, X. Yuan, L. Guo, and T. Liu, “Residual networks of residual networks: Multilevel residual networks,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 28, no. 6, pp. 1303–1314, 2017.