Determining the Efficacy of Simple Regression and Deep-Learning Models for Application to Resource-Constricted Classification Tasks

1st Johar, Kevin
Department of Computer Science. Western University.
London, Canada
ajohar5@uwo.ca

2nd Singh, Ashwin
Department of Computer Science. Western University.
London, Canada
asing465@uwo.ca

***Abstract*—Pretrained image classification models work effectively, yet are very large in size. For embedded and other resource-intensive applications, it then becomes necessary to use smaller models. Thus we aimed to determine whether neural networks or logistic regression models could effectively classify apparel data. Evaluation results then show that a simple convolutional neural network (CNN) with dropout and pooling layers provides a high validation accuracy of 90% at a size of 1.13MB. Hence, loading a smaller model is an effective alternative for low-memory uses. Further work is suggested in maintaining that level of performance and size but for more complex images.**

I. Introduction

Embedded applications must be reliable, fast, and secure. Therefore accessing cloud services for model inference is an ineffective solution since there could be network issues, timing delays in requests, and interception during transmission which could lead to stolen data [1]. Instead it is best to store the model on the machine and run inference tasks locally. It is possible to use pre-trained models for inference, yet the models require 16-548MB of space for use [2]. Embedded devices may support anywhere from a few kB to hundreds of MB in storage capacity [3]. Therefore, pre-trained models may be infeasible to use for an embedded application. Then the question becomes whether simpler models offer similar performance at a smaller size. Therefore the project aims to perform image classification using neural networks and logistic regression models to to check if the models could accurately be used in an embedded system.

II. Background & Related Work

A. Previous Attempts at Fashion Data Image Classification

Image classification for apparel data has been performed using large pre-trained networks including VGG-16, VGG-19, InceptionV3 and a custom CNN with millions of trainable parameters. The data used for evaluating each network included 30 categories of clothing items where each sample is represented using a triple-channel color image. Each image additionally is more organic as it contains the person wearing that clothing item [5]. From the experiment it was found that using pre-trained models offers better results than a custom CNN, yet the difference in performance is small. The data set used is more complex in the sense that each image is very different from another, therefore more parameters are required. Hence in theory, it could be possible to achieve high performance on a simpler data set using less parameters in a smaller CNN architecture. That is where research gap exists that we aim to fulfill by testing whether smaller models can work effectively for simple fashion apparel classification tasks.

B. Existing Developments in Resource-Constricted Learning

The following sections include existing work done to lower the computational burden in embedded applications. Methods to reduce the memory overhead and computation-load in resource-constricted applications include employing quantization and network pruning. Quantized deep neural networks attempt to use less bits to represent neurons and activation outputs. Challenges associated with this approach include difficulty with calculating gradients and ensuring proper gradient flow when very few bits are used for representation. Network pruning involves removing neurons from the network and reduce the complete number of computations performed. Methods for pruning revolve around removing neurons while tracking the change in the loss function to ensure network performance is conserved. It is evident that prior developments focus on modifying the computational graph itself or how it is encoded and stored at the hardware level. [4]

C. Activation Function Considerations

The following sections include neural network architecture considerations to maintain or improve performance while reducing computational overhead by layer selection.

Activation functions are applied to the output of each neuron to add non-linearity to the model. Without it the model can only learn linear behavior which may be unrepresentative of the data's underlying properties. Selecting the right activation function then becomes crucial. Activation functions must possess the following properties to be effective:

- Must be differentiable to ensure back propagation can proceed by application of the chain rule.
- Must prevent gradient from vanishing or exploding during loss function adjustment through back propagation. A vanishing gradient causes the network to converge very slowly due to small changes propagated to weights. Whereas an exploding gradient causes the network to possibly miss the optimal weight configuration due to large changes propagated to weights.
- Must have low computational cost.

Activation functions such as sigmoid and tanh result in a vanishing gradient, yet the following functions resolve that issue for both positive and negative vanishing gradients: leakyReLu, PReLu, ReLu6, SELU, Swish, hard-Swish and Mish. All aforementioned activation functions provide a transformed, continuous output and could be useful for the architecture of a CNN which involves lots of down and up sampling. Furthermore as tested by the authors, those activation functions work effectively for CNNs [6].

D. Layer Considerations

The pooling layer of the neural network ensures invariance to image transformations. If the image is rotated, translated or distorted, then the model becomes highly-sensitive to the noise and could perform ineffectively on validation sets. To counteract this effect, the added pooling layer down samples the image to reduce noisy behavior.

The dropout layer provides regularization to the neural network to prevent over-fitting. The parameters of other regularization methods include L1 and L2-regularization are computationally-expensive to tune. Therefore the dropout layer provides an efficient solution. It randomly selects a neuron to remove temporarily during training. The net effect is that the model learns a sparse representation of the input data rather than fitting to exact details. Hence the model can better generalize to unseen data [7].

III. Methods

A. Research Objectives

Research objectives for the study include:

1) Validation set accuracy by last epoch exceeds 90%
2) Model is at least less than 5MB in size, but ideally as small as possible
3) Model can be saved into file for easy use in embedded application
4) Model's f1 score exceeds 0.9 to ensure no false positives or negatives

Referencing the validation set for bench-marking model is necessary to check that the model is not over-fitting to training data. Furthermore, sizing requirements are put in place and later justified to ensure the model is small enough for use in an embedded application. Finally, the f1 score is essential as another performance measure to ensure that the model is not mis-classifying images often.

B. Research Methodology

Tested the hypothesis of whether smaller machine learning models perform simple image classification tasks effectively, where 'simple image classification' implies that the image contains just one subject. Furthermore, 'smaller' refers to memory size of the models themselves and any model less than 5MB in size is considered small enough. The selected threshold is based on the memory requirements in an embedded system, and on the size of the MobileNetV2 architecture which comes is the smallest pre-trained model coming in at a size of 14MB. Used Keras for implementing and testing the models, which is built on top of TensorFlow. TensorFlow uses 32-bit floating-point numbers to perform computations with, called the float32 data-type. Knowing this and the number of parameters stored in each model makes computing the model size in bytes a simple product between model parameters and the data-type size. Hence model size computed as follows: 4 * # of parameters.

Models that satisfied the requirements of being small and able to perform a simple image classification task include:

- Logistic Regression model
- Vanilla CNN model
- Modified CNN model

The vanilla CNN only contains convolution and fully-connected layers, furthermore it applies the commonly-used ReLu activation function. The modified CNN contains additional pooling and drop-out layers such that the model can learn a sparse representation of the data, be less prone to image transformations, and learn effectively with less parameters due to greater down-sampling. Additionally, the modified CNN architecture employs the use of a LeakyReLu activation function to allow better gradient flow during back propagation and improve training. The logistic regression model is the simplest one and involves only a sigmoid function tuned by an alpha parameter to perform classification.

Applied the models to the MNIST clothing data set which consists of 60000 training and 10000 validation examples of 28 x 28 gray-scale (single-channel) images. Decided to employ this data set due to its ease of setup as it is available as one of TensorFlow's built-in datasets, and it is sufficiently large.

The F1 score combines precision and recall together by computing the harmonic mean of the of two. Precision is a measure of the model's false positive rate and recall is a measure of the model's false negative rate. A higher precision and recall score indicate lower false positive and negative rates respectively. Given that the F1 score is a harmonic mean of them, the higher the F1 score is, the more samples are correctly classified. Evidently the F1 score combined with training accuracy paints a better picture of model performance, hence that is the metric used going forward.

$$F1(class = b) = \frac{2 * P(b) * R(b)}{P(b) + R(b)} \tag{1}$$

F1 score equation 1 where R(b) refers to the recall for class b and P(b) refers to the precision for class b.

$$R(class = b) = \frac{TP(b)}{TP(b) + FN(b)} \quad (2)$$

Recall equation 2 where TP(b) refers to all true-positives for class b and FN(b) refers to all false-negatives for class b.

$$P(class = b) = \frac{TP(b)}{TP(b) + FP(b)} \quad (3)$$

Precision equation 3 where TP(b) refers to all true-positives for class b and FP(b) refers to all false-positives for class b.

Evidently, the F1 score computation for a multi-class problem does not change, as it is in terms of a single class only. Hence can apply the formula individually across each class to determine the effective classification rate of every model.

C. Model Architectures

Model architectures for both neural networks to be tested are shown below. Evidently based on the parameters of each model, the modified CNN is approximately $\frac{1}{4}$ the size of the vanilla CNN, hence if it works effectively then it will show promising results for use in an embedded system.

D. Hyper-parameter Tuning

Before models were trained, needed to decide what value to use for the alpha parameter in the LeakyReLu activation function used in the modified CNN. The alpha value essentially dictates how large of a value to sub in place when the LeakyReLu function computes zero-values. Therefore the higher the alpha value is, greater values get outputted. Decided to tune the model on the entire training and validation sets to check which alpha value works best in the general case. Found that as the alpha value increases, performance drops off hence we decided to go with a smaller alpha value of 0.2, as indicated in figure 2.

IV. Experimental Results

Results obtained after training the models on test and validation sets are summarized. In figure 3, the F1-scores are compared to determine how many false negatives and positives the models produced. In figure 4, training and validation set accuracy is plotted across every epoch to see that the models do follow a pattern, converge, and do not over-fit to the data set. Furthermore in figure 4, the validation accuracy in the final epoch is compared against model size to determine which model maintains effective performs at the smallest size possible as per earlier mentioned research objectives. Computation for model size was based on the number of parameters in each model from the architecture diagrams shown in figure 1. Normalized confusion matrices used to show the true-positive classification rate across every class are shown in figure 5 for each model.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 conv2d_1 (Conv2D)           (None, 24, 24, 64)        18496

 conv2d_2 (Conv2D)           (None, 22, 22, 128)       73856

 conv2d_3 (Conv2D)           (None, 20, 20, 64)        73792

 conv2d_4 (Conv2D)           (None, 18, 18, 32)        18464

 flatten (Flatten)           (None, 10368)             0

 dense (Dense)               (None, 64)                663616

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 10)                330

=================================================================
Total params: 850,954
Trainable params: 850,954
Non-trainable params: 0
_________________________________________________________________
```

(a) Vanilla CNN

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)           (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_6 (Conv2D)           (None, 11, 11, 64)        18496

 conv2d_7 (Conv2D)           (None, 9, 9, 128)         73856

 conv2d_8 (Conv2D)           (None, 7, 7, 64)          73792

 conv2d_9 (Conv2D)           (None, 5, 5, 32)          18464

 flatten_1 (Flatten)         (None, 800)               0

 dense_3 (Dense)             (None, 128)               102528

 dense_4 (Dense)             (None, 64)                8256

 dense_5 (Dense)             (None, 32)                2080

 dropout (Dropout)           (None, 32)                0

 dense_6 (Dense)             (None, 10)                330

=================================================================
Total params: 298,122
Trainable params: 298,122
Non-trainable params: 0
_________________________________________________________________
```

(b) Modified CNN

Fig. 1: CNN Architectures and Parameters

V. Discussion

It was found that model performance across both the vanilla CNN and modified CNN remained similar, both in terms of their validation accuracies and F1-scores. It implies that even with less parameters, performance can be maintained for a simple image classification task. In particular, the high F1-score is especially great since it means that the model correctly classifies images most of the time. Furthermore, the high validation accuracy scores are promising because it means that

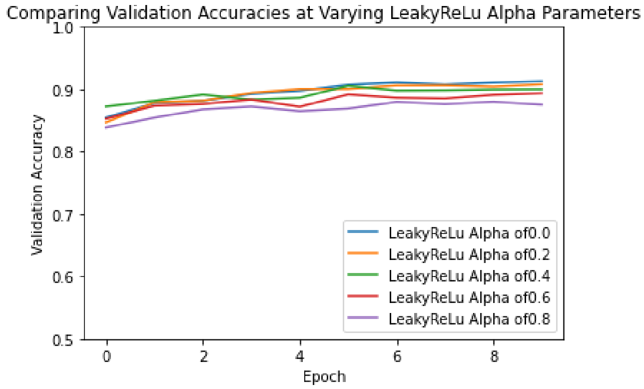


Fig. 2: Modified CNN Activation Function Tuning

F1-Score Comparison Across Models		
Vanilla CNN	Modified CNN	Logistic Regression
0.90567	0.90526	0.432

Fig. 3: F1-score Comparison

the models do not over fit to the training set. With the use of pooling and drop-out layers of the modified CNN, it should have learned a transform-invariant, sparse representation of the data set hence it should work better for more abstract, complex images.

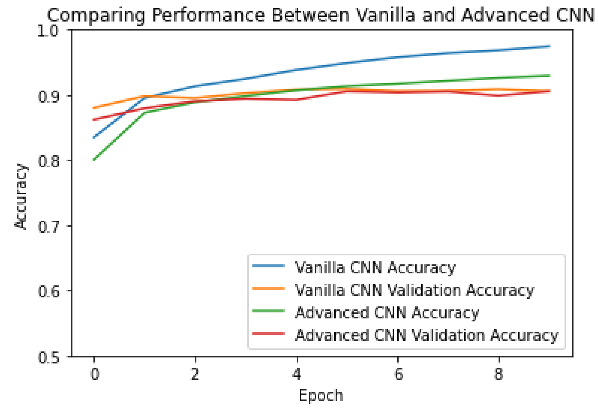
To gain a higher level of resolution into the classification rate for each class in every model, can refer to the confusion matrices. It is evident that both CNN models suffered on classifying image examples from class 7 which includes images of sneakers. Looking at the sneakers there is a lot of variation in the silhouette and details, which could have been a problem area for the models. Evidently the models are still very limited given the images are all low resolution and exclude many details. Model performance is undermined given the simplicity of images from other classes. If more complex images were to be used, or less structured images with more things in the background, then model performance would definitely suffer.

Overall the modified CNN maintains a validation accuracy of approximately 90% with an F1-score of 0.905 all while maintaining a size of 1.13MB. Hence it is evident that smaller models can effectively work for simple classification tasks in a resource-intensive system. However further work must be done to validate results for more complex image data sets.

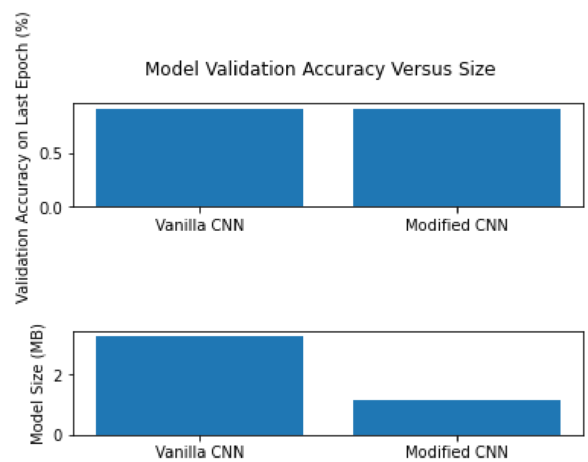
VI. Conclusions and Future Work

It is evident that either a Vanilla CNN or a more advanced CNN including dropout and pooling layers work effectively to perform image classification on small, simple images. It is evident that using dropout and pooling layers can down-sample images greatly and simplify model parameters to achieve massive reductions in memory usage, which is crucial for resource-intensive applications. Hence simpler deep-learning models can be built in-house for embedded devices rather than using pre-trained off-the-shelve networks such as VGG-19.

For the future, additional work can be done using more complex datasets of higher resolution images to check if simple models can still be used as an alternative to pre-trained very deep neural networks.



(a) Training and Validation Performance



(b) Size Versus Accuracy Comparison

Fig. 4: Model Performance Comparison

References

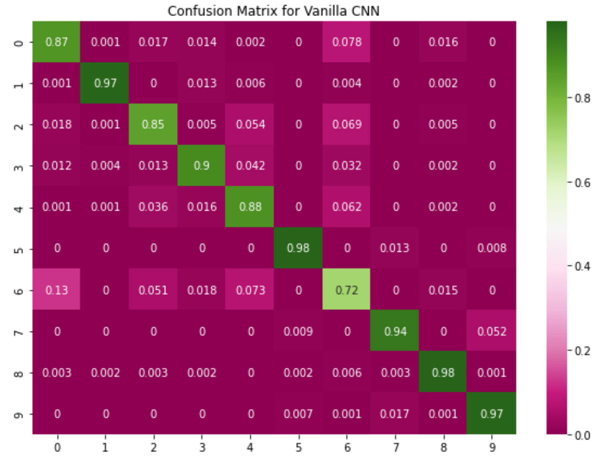
[1] Nadeski, M. (2019). *"Bringing Machine Learning to Embedded Systems,"* Texas Instruments. https://www.ti.com/lit/wp/sway020a/sway020a.pdf?ts=1647453459210&ref_url=https%253A%252F%252Fwww.google.com%252F

[2] Keras. (2022). *"Keras Applications,"* Keras API Reference. https://keras.io/api/applications/

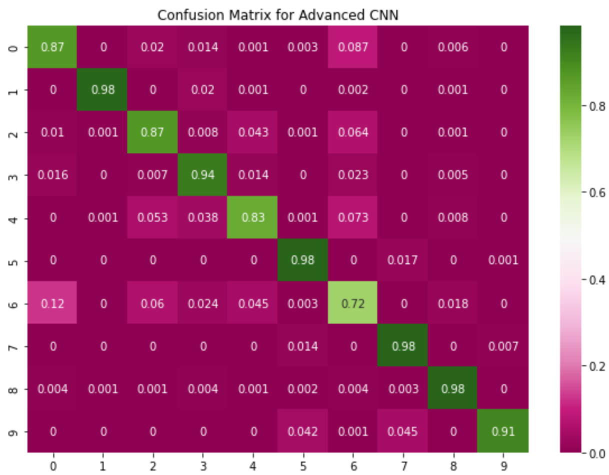
[3] N/A. *"Embedded Computer Systems,"* Alness Gnomio. https://alness.gnomio.com/pluginfile.php/209/mod_resource/content/1/On-line%20Resources/C%20Systems%20Int2/page_26.htm

[4] Roth, W., Schindler, G., Zohrer, M., Pfeifenberger, L., Peharz, R., Tschiatschek, S., Froning, H., Pernkopf, F., Gahramani, Z. (2020, January 7). "Resource-Efficient Neural Networks for Embedded Systems," doi: 10.48550/arXiv.2001.03048.

[5] Schindler, A., Lidy, T., Karner, S., MonStyle, M.H. (2018, November 11). "Fashion and Apparel Classification Using Convolutional Neural Networks," Proceedings of the 10th Forum Media Technology and 3rd All Around Audio Symposium, St. Poelten, Austria, November 29-30, 2017. https://arxiv.org/abs/1811.04374



(a) Vanilla CNN Confusion Matrix



(b) Modified CNN Confusion Matrix

Fig. 5: Analyzing Classification Results By Class

https://alness.gnomio.com/pluginfile.php/209/mod_resource/content/1/On-line%20Resources/C%20Systems%20Int2/page_26.htm

[6] W. Hao, W. Yizhou, L. Yaqin and S. Zhili, "The Role of Activation Function in CNN," 2020 2nd International Conference on Information Technology and Computer Application (ITCA), 2020, pp. 429-432, doi: 10.1109/ITCA52113.2020.00096.

[7] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhudinov, R., "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," Journal of Machine Learning (2014), 1929-1958. https://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf