COMP 1801: Machine Learning

Coursework

Joshua Roses-Agoro
000992718
University of Greenwich
MSc Data Science

***Abstract*—This report contains an investigation into machine learning algorithms, more specifically, image classification models created by usage of sequential keras models that have been implemented in python. The purpose of this investigation is to demonstrate thorough understanding of the full pipeline associated with obtaining a dataset, pre-processing said data, defining model constructs, training a model, and fine-tuning it. Before it can be evaluated and fully understood.**

Keywords—Machine Learning, image classification, SGD, RMSProp, fashion_mnist, Keras, augmentation.

I. Introduction

Machine Learning is an important part of modern-day society, playing a key role in our development and further advancements into a more technologically enabled era. The focus of this coursework is drawn toward a subset of Machine Learning known as image classification and the full pipeline associated with it which is a complex subject with many potential avenues, but the one being taken in this instance is entirely to do with Machine Learning image classification models, and how they can be trained and evaluated in the most accurate and efficient manner.

The first step in this process is choosing an appropriate dataset which can be used to demonstrate the above, in this instance the dataset chosen is to do with image classification and it is publicly available. Two different models have been trained and evaluated according to this dataset, and in doing so prove what a good model should look like, and what a bad model looks like. By thoroughly demonstrating this process, the steps needed to successfully implement a well fitted model are made apparent in order to create a greater understanding of how models should be prepared, trained, evaluated and fine-tuned.

Particular attention needs to be drawn toward the goals and outcomes of this coursework, the pipeline otherwise known as the steps required from obtaining and preparing a dataset, all the way to fine-tuning the finished product are what the core focus is drawn toward, for the best possible comprehension. And although a minimum accuracy of 50% must be achieved on the model, the aforementioned pipeline being implemented correctly is of utmost importance, before even taking the accuracy of the model into consideration.

II. Related Work

Building powerful image classification models using very little data is a paper provided by Francois Chollet and he belongs to the Keras Developer community which are responsible for a substantial amount of information and data on the subject of artificial neural networks and this particular work closely coincides with what has been attempted and achieved in this coursework.

To achieve the highest possible accuracy with a trained model, typically a large dataset is needed which contains what is needed for both testing and training your model.

This paper discusses the ways in which a small network can be created from scratch using the bottleneck features of a pre-trained network, and finally fine-tuning the top layers of the previously mentioned network. The steps taken throughout this paper are very similar to that of what has been achieved in this coursework, except the model in question from the paper uses a significantly smaller dataset, thus demanding a more refined and tuned model if a high accuracy and low loss is expected to be achieved, because there isn't a substantial amount of data to use for both training and validation.

The first step taken in this scenario is to do with data-preprocessing in the form of data augmentation. What this means is that the training examples have been augmented against a number of random transformations, and this ensures that the model never uses the same exact picture twice and this is vital in ensuring the prevention of overfitting and in doing so, the ability for the model to generalize increases.

Once the function is implemented which augments the data, a tool is used to generate a few images from the dataset and save them to a temporary directory in order to create a better understanding for what the augmentation strategy is doing once implemented. With such a small dataset, as mentioned previously, a large problem is overfitting, and augmentation alone isn't enough to prevent this from happening. As a result, focus is drawn toward the entropic capacity of the model which in layman's terms is how much information the model can store. When looking at what this would mean in a real world scenario, we realize that models which are allowed to store more information, also have a higher potential at greater accuracy, but, more data also equates to a large possibility for irrelevant features to be stored which can lead to inaccurate models.

Thus, ensuring that models which are only capable of storing fewer features are forced to focus on the most significant features within the dataset. This significantly contributes to the relevancy of the data being used which ultimately leads to higher accuracy.

The method used to modulate entropic capacity in this instance is the choice of number of parameters within the model, which comes down to how many layers there are and the corresponding size of each. A small number of layers and filters were used in this case followed by ReLu activation.

After 50 epochs, this approach immediately achieves a validation accuracy of between 0.79 and 0.81 before any major fine-tuning has been done, which is impressive. The next step is to implement bottleneck features which yielded an accuracy of 0.90 in a minute, but this is only necessary when trying to achieve the highest accuracy possible with a small dataset which doesn't have sufficient data for substantial training so the model can fit the data properly.

Fortunately, it is not needed in the scenario relating to the coursework task because a dataset of our choice could be used, thus allowing for sufficient data samples to be provided.

Fine-tuning can only occur on a successfully trained model because all layers need to have properly trained weights in order to further optimize them. The top-level classifier is trained first, followed by the tuning of the convolutional weights alongside it. It is done in precisely this way to prevent overfitting, because it is commonly known that the entire network has a large entropic capacity and so in the prevention of overfitting the network cannot be fine-tuned as a whole.

The final step in fine-tuning is using a very slow learning rate for the model. This is best done incrementally with the SGD optimizer instead of RMSProp which is an adaptive learning optimizer which has large increments in its updates and that has the potential to completely eradicate any previously learned features which may be of importance whilst perfecting the model.

III. Dataset Preparation

The first step in this section was to choose an appropriate dataset, in this case the fashion_mnist dataset was chosen because it is beginner friendly and perfect for a basic sequential model that could be used for image classification.

Tensorflow has a data fetching API that makes the process of importing a dataset extremely simple. How this works is by creating parameters which the data can be extracted into and then defining the boundaries of said parameters with an appropriate type to house what the dataset contains.

Train set:	
Sizes:	60000, 28, 28
Test set:	
Sizes:	10000, 28, 28
Image values: min/max	0 to 255
Label values: min/max	0 to 9

As we can see from the above table, for the train set there are 60,000 images with a size of 28x28 which means they have a length and width of 28 by 28 pixels. And for the test set there are 10,000 images with the same dimensions. The maximum image values will be used later to normalize the dataset.

What needed to be done next is the normalization of the dataset which is an important phase of pre-processing before the data is ready for the model. Firstly, the raw image data for both the test and train set can be normalized by dividing each by the maximum value, which is what we identified after downloading the data and extracting it into a print statement within confined bounds.

Next the labels pertaining to each image within the dataset needed to be converted to one-hot vectors, what this does is allow categorical data to be understood by a Machine Learning algorithm because by default, many algorithms cannot interpret categorical data in its raw form, and so by converting the label to a one-hot vector, the categories are being converted into numbers and in doing so, allowing the algorithm to interpret it in an understandable manner for processing.

IV. Proposal

For this coursework, the most appropriate model to be used was the sequential model. It is very well suited to a plain stack of layers, where each layer has exactly one input tensor and one output tensor, which is exactly what we are dealing with in this specific scenario.

In order to achieve a successful sequential model, what firstly needs to be done is the passing of list layers to the sequential constructor which is a function built specifically for Keras models. The layer needs to know the shapes of their inputs so that the weights can be created and from there a model compiler can be added with an optimizer which determines the learning rate of the model itself. Typically, a good model will consist of a very slow learning rate rather than an adaptive learning rate optimizer because this ensures that the magnitude of the updates stay very small and in doing so ensures that previously learned features are not totally overwritten or forgotten.

For the purpose of explicitly demonstrating the differences between a good and bad model, both have been implemented in this coursework so that they can be contrasted with one another.

When defining the model constructs, with the good model there are multiple layers used beside of the input dimension and output dimension, and these layers add dropout and dense. The dropout layer sets input layers at random to 0 with a frequency of whatever rate you set it to be, which aids in the prevention of overfitting. And the dense function determines how many neurons are present in the first hidden layer.

With the good model multiple layers had been set making use of these functions to prevent overfitting from occurring which ensures a more accurate model, whereas with the bad model there are only two layers, one for the input dimensions, and one for the output dimensions, which is the absolutely bare minimum that the model may have.

Next, when compiling a model an optimizer needs to be added, there are several options which exist for numerous reasons, but they all have a series of pros and cons depending on how they are being implemented. For the good model, the Stochastic Gradient Descent (SGD) optimizer was used because it offers an iterative method for optimizing which is also customizable in the sense that a custom learning rate can be set, and this is crucial in ensuring that the model learns slowly over many iterations so that the previously learned features are not wrecked, as was mentioned previously.

In the case of the bad model, the RMSProp optimizer was used which is an adaptive learning rate optimizer. And as previously discussed, this method of optimization can be dangerous because updating the model occurs in large magnitudes and it can completely destroy previously learned features which may be beneficial to the accuracy of the model.

When it came to the training of the actual models, the two most important factors to take into consideration are the number of epochs, and the validation split.

With the good model 20 epochs were set with a validation of 0.06 which means that out of 60,000 samples, 10,000 would be set aside for the purpose of validation, and this is vital in ensuring that the model which was trained is actually accurate in a non-superficial way.

The bad model was only given 3 epochs with a validation split of 0.01 and what this means is that there is no way the model can fit correctly due to an overall lack of training which is largely related to how many epochs were set, and also a miniscule amount of data was set aside for validation, resulting in the model creating inaccurate results, and then unsuccessfully validating them because there aren't enough samples to check against. Thus, the model couldn't make changes to fit the data properly.

V. Evaluation

With the above methods described now fully implemented, the results from both models can now be contrasted with one another. With the pros and cons of each approach highlighted.

The good model achieved extremely satisfactory results with a loss of 0.35 and an accuracy of 0.87. Those numbers immediately look good and are backed up by the plots. As we can see from both the loss plot and accuracy plots which can both be found in the appendix as figure 1 and 2, the training line is extremely close to the validation, which notifies us that the model has successfully fit the data.

A good fit was the aim of this coursework from this beginning and is the goal of any learning algorithm which exists between the overfit and underfit models.

"A good fit is identified by a training and validation loss that decreases to a point of stability with a minimal gap between the two final loss values.

The loss of the model will almost always be lower on the training dataset than the validation dataset. This means that we should expect some gap between the train and validation loss learning curves. This gap is referred to as the generalization gap." - (Brownlee, 2019)

The bad model, otherwise known as the unsuccessful model, can be deceiving if not understood correctly. When only taking into consideration the final results of the trained model which can be found in figure 7 of the appendix, we see a loss of 0.61 and an accuracy of 0.88. To the uninformed eye this may appear to be a good result, but that is far from the truth.

In figure 5, we can see a plot which explains what happened during the training process and we can see a massive overfit of the data, and there are two primary reasons why this occurred.

The first of which being due to the lack of training that was done. Only 3 epochs are not enough to train a model with a dataset of this side and a model typically needs many iterations to train accurately and fit the data correctly.

The second reason is due to the miniscule amount of data that was set aside for validation, thus resulting in a model which can not effectively train according to the available data, and then misinterpreting the accuracy because it isn't able to validate correctly.

There are several pros of the successful model and they are listed as follows: Several layers were added in the model constructs before training in order to correctly establish the weights. An SGD optimizer with a very slow learning rate was also added to ensure small increments would take place whilst training occurs as to not overwrite previously learnt steps, and finally, a fairly large number of epochs were used in the training process with a sizeable number of samples set aside for validation. Thus, ensuring that a model that seemingly appears to be a good fit, can successfully be validated.

There are several cons of the unsuccessful model and they are listed as follows: Only two layers were defined within the model constructs and it was merely the input dimension and the output dimension. The RMSProp which is an adaptive learning optimizer was used and this has a large magnitude in the sizes of increments during training, which most likely eradicated important features learnt during training. And finally, a finite number of epochs were used for training which clearly wasn't enough for the model to successfully fit the data, and a ridiculously small number of samples was set aside for validation. Ensuring that whatever the model was able to learn, could not even be validated correctly.

VI. Future Work

There are several avenues which can be explored in order to extend the work accomplished in this coursework. But there are two which should primarily be focused on.

The first being how to further fine-tune a model which will inevitably increase the accuracy and decrease the loss as progression is made. By adding more layers to the model constructs with different, more finite parameters which are more appropriately adjusted to the weights. And by potentially using an alternate learning rate optimizer which may offer more benefits and a slower learning rate which is entirely in favor of the model. And lastly the training process could be extended too, by increasing the number of epochs this is almost a guarantee that the trained model will be more accurate because it has more time to fit the data correctly. And in relation to the number of epochs, a bigger sample set of data could be placed aside for validation because this may ensure a higher accuracy too.

The second avenue worth exploring would be into different models. The sequential model approach was used for this coursework as this is what was taught throughout the semester, but this doesn't make it the best and most optimal approach. It is possible to construct a model using API building and this is less time consuming and more efficient for starters, which grants extra time for optimization in various different forms.

VII. References

[1] Mayo, M., 2018. Building A Basic Keras Neural Network Sequential Model - Kdnuggets. [online] KDnuggets. Available at: <https://www.kdnuggets.com/2018/06/basic-keras-neural-network-sequential-model.html> [Accessed 7 December 2020].

[2] Brownlee, J., 2019. How To Use Learning Curves To Diagnose Machine Learning Model Performance. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/> [Accessed 8 December 2020].

[3] Chollet, F., 2016. Building Powerful Image Classification Models Using Very Little Data. [online] Blog.keras.io. Available at: <https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html> [Accessed 8 December 2020].

[4] Falbel, D., n.d. Guide To The Sequential Model. [online] Keras.rstudio.com. Available at: <https://keras.rstudio.com/articles/sequential_model.html> [Accessed 9 December 2020].

[5] Team, K., n.d. Keras Documentation: Layer Activation Functions. [online] Keras.io. Available at: <https://keras.io/api/layers/activations/> [Accessed 10 December 2020].

[6] Team, K., n.d. Keras Documentation: Optimizers. [online] Keras.io. Available at: <https://keras.io/api/optimizers/> [Accessed 10 December 2020].

[7] Polceanu, M. and Suzuki, A., 2020. Neural Networks. [online] Moodlecurrent.gre.ac.uk. Available at: <https://moodlecurrent.gre.ac.uk/course/view.php?id=61075> [Accessed 10 December 2020].

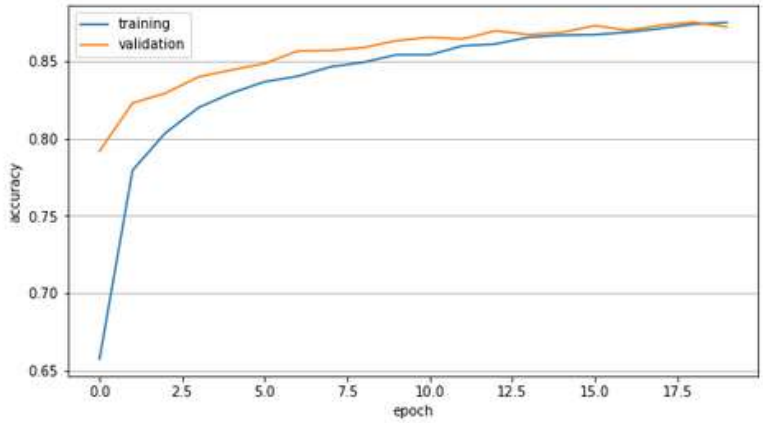

Figure 1 – Good model Accuracy

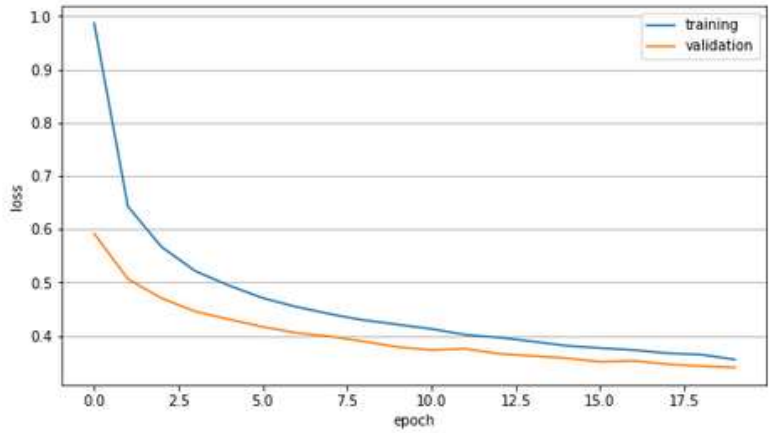

Figure 2 – Good model Loss

```
Test loss: 0.35082176327705383
Test accuracy: 0.8736000061035156
```

Figure 3 – Good model Loss/Accuracy

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_29 (Dense)             multiple                  615440
_________________________________________________________________
dropout_20 (Dropout)         multiple                  0
_________________________________________________________________
dense_30 (Dense)             multiple                  78500
_________________________________________________________________
dropout_21 (Dropout)         multiple                  0
_________________________________________________________________
dense_31 (Dense)             multiple                  1010
=================================================================
Total params: 694,950
Trainable params: 694,950
Non-trainable params: 0
```

Figure 4 – Good model Summary

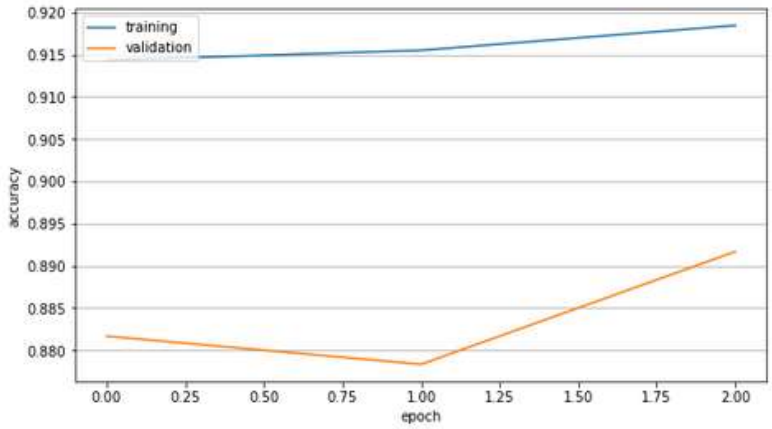

Figure 5 – Bad model Accuracy

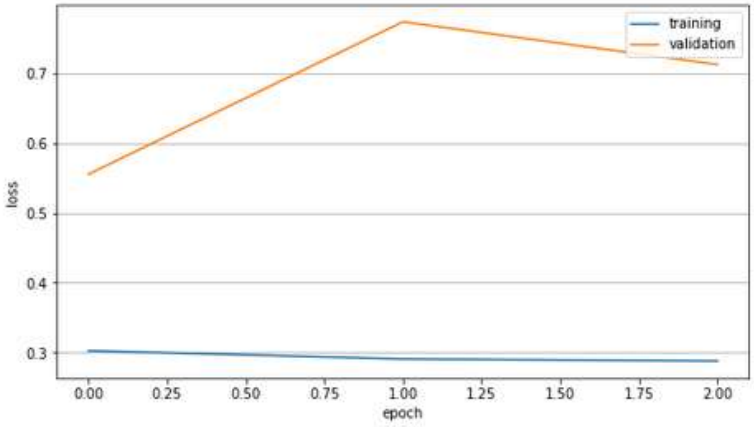

Figure 6 – Bad model Loss

```
Test loss: 0.6140872836112976
Test accuracy: 0.8805999755859375
```

Figure 7 – Bad model Loss/Accuracy

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_12 (Dense)             multiple                  615440
_________________________________________________________________
dense_13 (Dense)             multiple                  7850
=================================================================
Total params: 623,290
Trainable params: 623,290
Non-trainable params: 0
```

Figure 8 – Bad model Summary