Computer Vision using Deep Learning models

An analysis of three Deep Learning models using MNIST database

Nikhil Mathews
Master's in Computer Science
University at Albany
Albany, New York

***Abstract*—An analysis of three deep learning methods in the realm of computer vision. This includes Artificial Neural Network (ANN), Recurrent Neural Network (RNN), and Convolutional Neural Network (CNN). The dataset used for the analysis is MNIST. Each model is created based on the above mentioned methods and trained using the original MNIST dataset, or after it has undergone some sort of preprocessing. The behavior of each of them are tested using data not encountered before during training, such as a noisy test set, and the observations noted. The response of these models to pure noise data are also going to be recorded. We will attempt to look for bias in predictions for models trained with various combinations of training and test data sets. Finally, we will come to some conclusions regarding the properties of each of these Deep Learning methods with respect to the MNIST dataset.**

I. Introduction

Deep learning is part of a family of machine learning methods based on artificial neural networks that mimics the workings of the human brain in processing data for applications such as speech recognition and translation, decision making, object detection, etc. One major use is in the field of Computer Vision whose applications vary from face recognition to self driving cars. In this scenario, we will examine its properties and applications in the MNIST data set, which is a large database of handwritten digits widely used for training and testing in the field of machine learning. This is because it is simple to manipulate and implement owing to the fact that it is 2-dimensional unlike other images which have to account for R,G,B. Also, given the fact that it's pixel density is 28×28, the computational power required is far lesser making it viable for testing. Each image has a resolution of 28×28 with values ranging from 0 to 255, with 0 used to represent black and 255 for white. We divide each of the pixel values by 255 to normalize the input data because this speeds up learning and leads to faster convergence in our deep learning models. We get our MNIST data set from keras where the digits are white and the background is black. It has 60,000 images for training and 10,000 for testing.

Fig. 1. Handwritten digits from the MNIST data set.

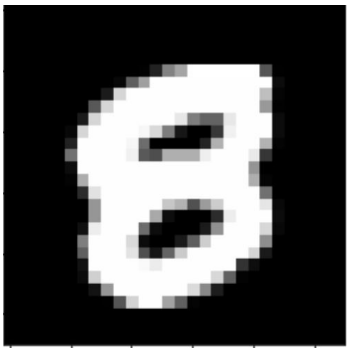


Fig. 2. Original MNIST data

II. Tools Employed

The Deep learning methods used are as follows:

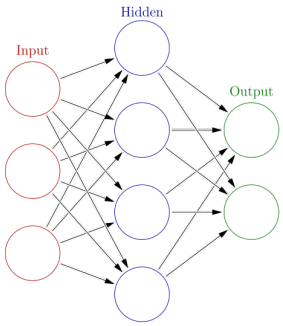
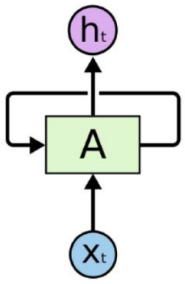


Fig. 3. Artificial Neural Network

1) **Artificial Neural Network (ANN)** [1] :
The most simplest of Neural Networks, loosely modeled on the neurons in a biological brain, this still maintains a powerful hold in the world of deep learning for its versatility and simplicity in understanding and

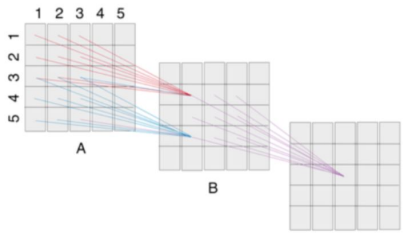
implementation. They receive inputs from neurons before it, calculates a number based on the values of the input it receives and the respective weights for each input, as well as a bias value, and send an output that is a function of that number. We will flatten the 28×28 MNIST image into a 784×1 vector each of which is sent as a feature to the input of the ANN. In supervised learning, which is what happens here, the error in the predicted value is conveyed via backpropagation to the neural layers before it causing them to adjust weights accordingly, which usually happens through gradient descent. In our case, the model will make the input go through three hidden layers of 128 neurons each with a 20% dropout. The activation function used will be *relu* to avoid the vanishing gradient problem.

2) **Recurrent Neural Network (RNN)** [2] :



Based on the ANN architecture, they are a class of neural networks that allow previous outputs to be used as inputs while having hidden states that makes it ideal for prediction of temporal data. At a high level, it remembers the past and makes predictions based on what it learnt from the past. The Long short-term memory (LSTM) is based on the RNN architecture uses a combination of various gates to retain context during predictions and so is the most popular. In this case, the MNIST image is pigeonholed as an sequence into the RNN as an input. In our implementation, the LSTM will have 128 hidden neurons with a 20% dropout. A GlobalMaxPool1D layer is used to get the highest among the hidden layers before it is connected to the output layer.

3) **Convolutional Neural Network (CNN)** [3] :



We start with a standard input like for the ANN, then begin feature extraction which is done using several pairs of Convolution layers or Kernels, and pooling filters. The convolution operation extracts high-level features such as edges, from the input image after which the pooling layer reduces its spatial size to decrease the computational power required to process the data through dimensionality reduction. It also helps us find dominant features. CNN is one of the most widely used Deep Learning methods in computer vision. In our case, the input will have to first be converted in a 3D format since CNN processes image for RGB data. Since our data is black and white, each image can be represented as $(28, 28, 1)$. A Keras Conv2D layer is employed that uses 256 filters and a $3 * 3$ kernel with a relu activation. A Maxpooling of size $2 * 2$ is carried out before the values are flattened and connected to a 64 size dense layer before it is finally sent to the output layer. Most layers described above have a 20% dropout.

In each case, the output layer will be equal to the number of labels (which is 10), the result will go through a softmax function and the label with highest value will be declared as the predicted result. The loss therefore is evaluated using sparse categorical cross entropy as the cost function when the Adam optimizer makes corrections during backpropagation.

III. Methodology

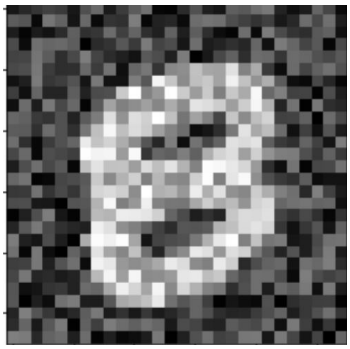
Now that we have established our deep learning networks, the next task is to examine the behavior of each of them using MNIST data sets of various configurations. First we go the conventional route and examine the effects of each model on standard MNIST data by emphasizing on testing data prediction which is usually carried out during validation. We get to see graphically the change in evaluating testing data during each training epoch. We also compare the prediction values with actual ones to see if there are some numbers the model has difficulty with, or conversely to look for prediction bias. Keep in mind that the output layer represents each label whose value is determined by the softmax activation function. This gives us a 'confidence distribution' or in other words, the opportunity to see how a model 'sees' a data and how sure it is about its prediction.

One factor we will analyze will be intelligence which can be described in 2 ways. First, it is the ability to extrapolate or predict with less information. In our case, we will see how well a model can predict test data given limited training data. Another way to look at intelligence is in context of versatility: how well can an agent apply his learned experiences to an entirely different set of problems. For our case, we will train the three models and expose them to a different set of data to make predictions. This different set of data is none other than MNIST data that has been manipulated in several ways as shown below.

We can create a noisy test data which is done using

$$Data.apply(lambda\ x : x - random.rand())$$

and gives us data that looks like this:



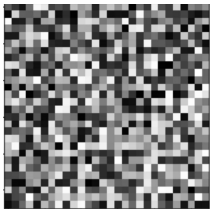
We will see how accurately each model predicts this kind of data. Then, each of the models will also be trained using this noisy data and we will see if this makes a difference in predicting noisy test data as well as the original test data. An analysis will be made regarding mistakes made by each model to see if this was something that could be forgiven, i.e, if these errors could be made by humans as well.

We will create an inverted test data shown above which is created using:

$$Data.apply(lambda\ x : 1 - x))$$

and see how this is perceived by each of the models using all the methods mentioned above.

Next, we will create pure noise data that looks like this:

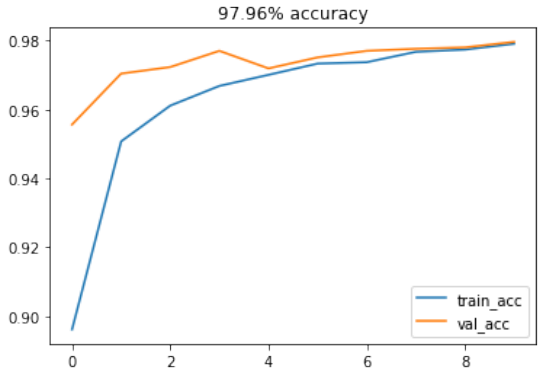


Then, again, we will see how each model perceives them. During the analysis, we will look at the predicted labels that each of the models gives us for a set of noise input and see if it perceives them as any number in particular. We will also examine the confidence with which each model predicts noise data as a particular label. Ideally, we should see noise data being categorized as equal probability/value for all labels. But that will soon be shown as not the case. We will also see how confidently these models label a noise data with the threshold set to 90% and described in this paper as 'high confidence'. At every step, this information will be displayed graphically so that the reader can grasp the developments with ease.

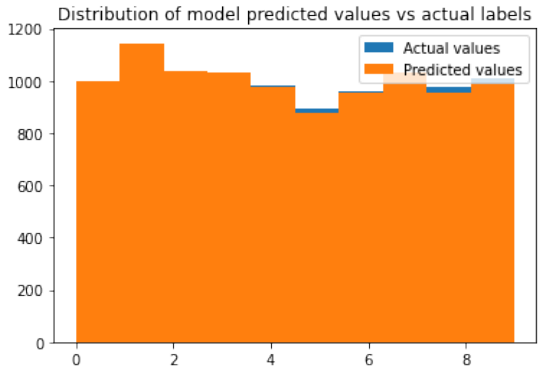
IV. Results

A. Artificial Neural Network (ANN)

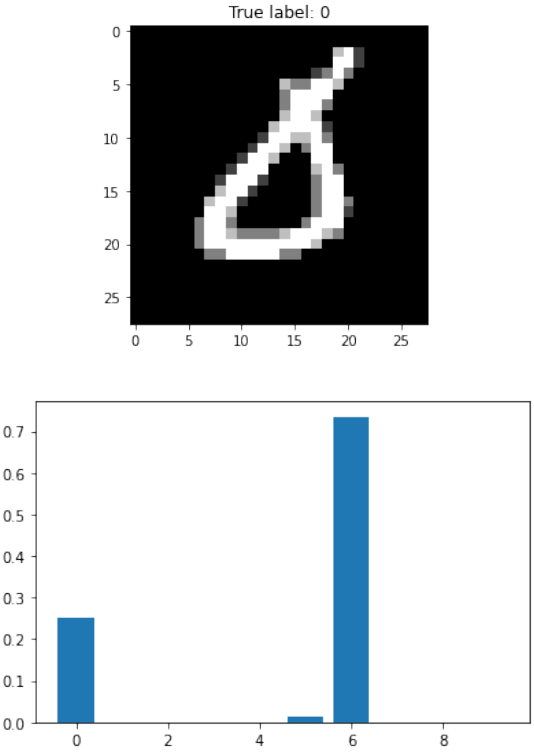
The simplest and most standard approach for MNIST training, this is done in 10 epochs and gives us a high testing data accuracy of 98%.



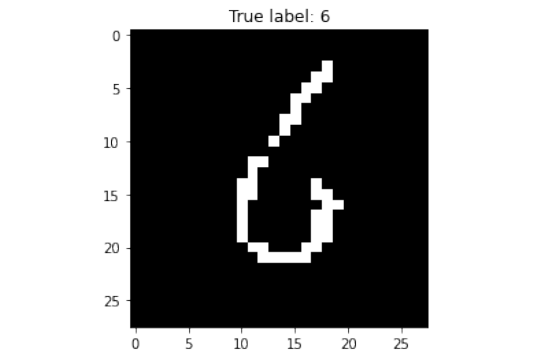
Most predictions were correct with the model unable to predict 5, 8 and 9 to a small extent.



Most mistakes made here seem forgivable for instance the image below was predicted as 6, something that most humans would do.But when we look at the confidence distribution, we see that it had considered the possibility of zero, something again, that most people would do.



This by itself should prove how adept the simple multi layer perceptron is when it comes to recognizing hand written digits. When we evaluate for intelligence by reducing the number of training data to 1000 while keeping the same testing data as validation data, we get an accuracy of 86.5%. Now, if we do a preprocessing where every value in the input matrix above 90% of the pixel value is changed to 100% while other pixels are at 0%, for example like this: and see if this makes a



difference in intelligence, it does not. The validation accuracy stays almost the same.

The ANN model is now trained on normal data and evaluated on noisy data, and gives the following results as shown in Fig.4. We see that it cannot get high accuracy and just gets the answer right half the time. On the other hand, when we train with noisy data, as shown in Fig.5, we get a much better performance. The prediction distribution is shown in Fig.6. It shows the model predicts 1,4 and 9 more often while not recognizing 3,5 and 8. For eg, a 3 as shown in Fig 7. was labeled as a 2.

Fig. 4. ANN trained on normal, evaluated on noisy data

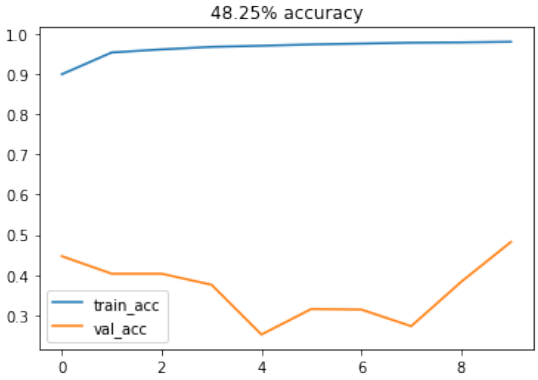


Fig. 5. ANN trained and evaluated on noisy

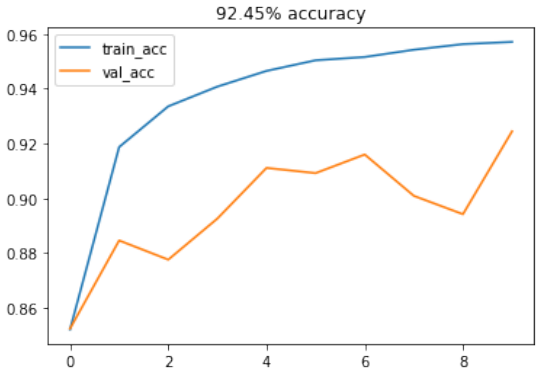
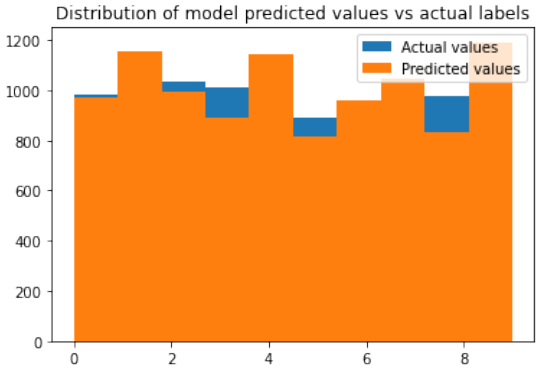


Fig. 6. ANN trained and evaluated on noisy: prediction distribution



When we look at the confidence distribution, we see that 3 was the models second guess.

Fig. 7. ANN trained and evaluated on noisy: predicts as 2

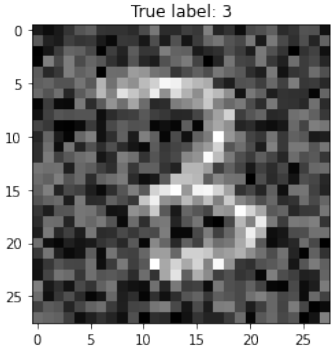
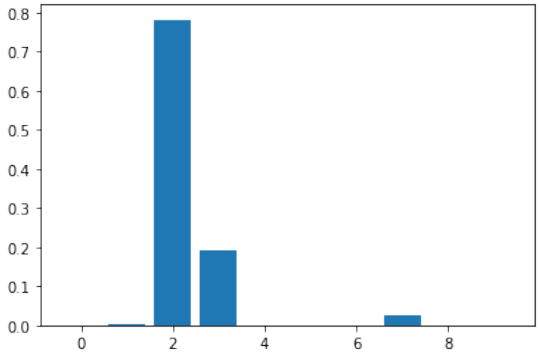
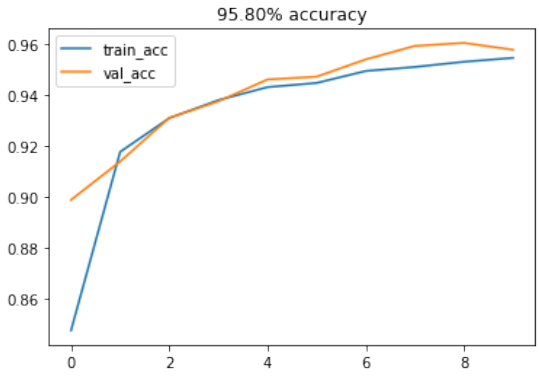


Fig. 8. prediction distribution of 3



Now what if this model that is trained on noisy data is exposed to normal data? It seems to give good accuracy of 96% for the normal test data, as shown in Fig.9, which is almost as high as when it was trained on normal data alone, making it more versatile and therefore, more 'intelligent'. What this means is that the ANN trained on noisy data alone should give good performance on either case. Another interesting fact to note is that the model trained on noisy data gives better results for normal data.

Fig. 9. ANN trained on noisy and evaluated on normal data



The model failed to correctly label several numbers and was biased towards 1 and 8. One of the mistakes made by the model is the number 4 predicted as 7 as shown in Fig 10. According to the prediction distribution, 4 was its next choice.

Fig. 10. ANN trained on noisy and evaluated on normal: predicts as 7

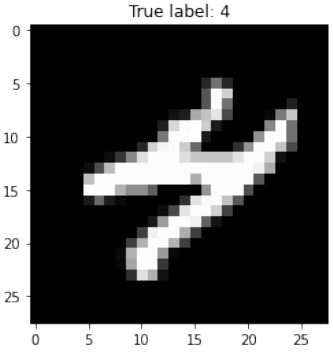
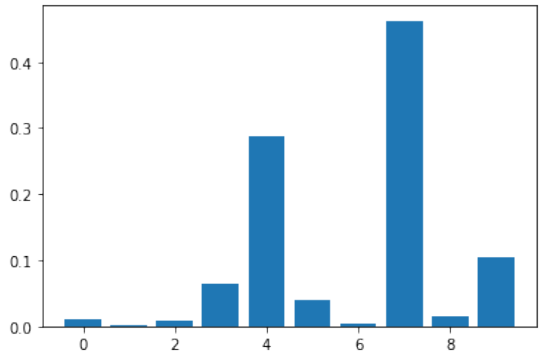
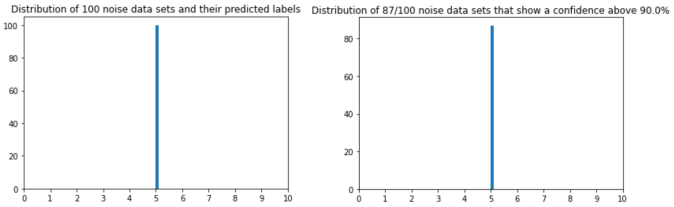


Fig. 11. prediction distribution of 4



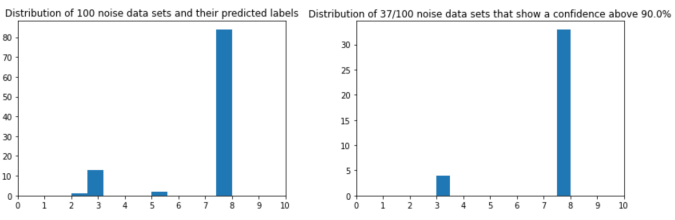
Now we will look at how the ANN model trained on normal data reacts to pure noise. A set of 100 pure noise images are sent as input and their predictions are as follows. Oddly, it

Fig. 12. Noise analysis of ANN model trained on normal data

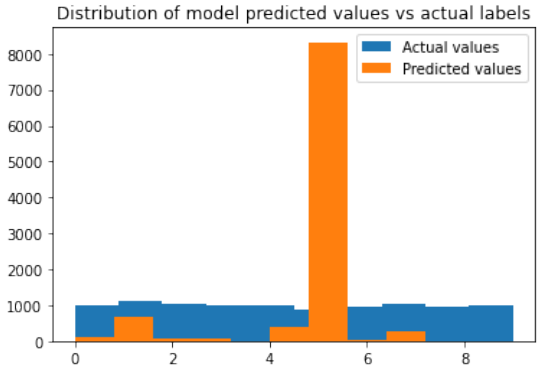


predicts all of them as 5 and most of them are labelled so with high confidence. On the other hand, ANN model trained on noisy data sees most noise as 8 and 3 with just 37% of them labeled with high confidence.

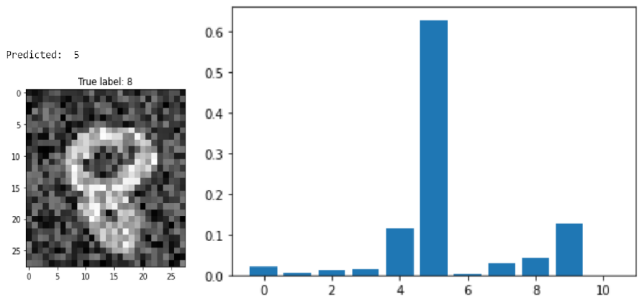
Fig. 13. Noise analysis of ANN model trained on noisy data



We will now try something new and one off. 5000 pure noise datasets are created and added to normal training data and labelled as an eleventh label: 10. 1000 of them are added to normal test data. When the model is trained on this, it gives an accuracy of 98% and identifies pure noise correctly and with high confidence. We can also create an ANN that is trained on noisy data and pure noise and it does a good job of labelling original data and noise correctly at 98%. But it does a bad job of labelling noisy test data with most predictions being 5 and giving an accuracy of 19% as shown below.

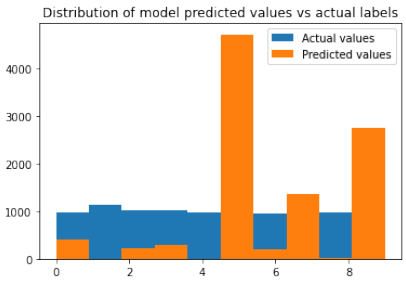


This also seem to shows high confidence in its incorrectly predicted labels. For example, as shown below, an 8 is predicted as 5.



We see that the correct answer is not even in the model's top 3 guesses. This makes it very non-intuitive compared to the previous ANN models we used. When either of these models are faced with inverted data, the accuracy is around 0% since most of them are seen as label 10: pure noise.

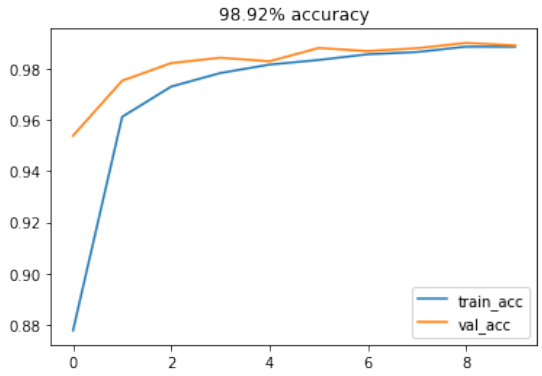
When standard ANN trained on normal data is exposed to inverted data, predictions vary but 5 seems dominant.



B. Recurrent Neural Network (RNN)

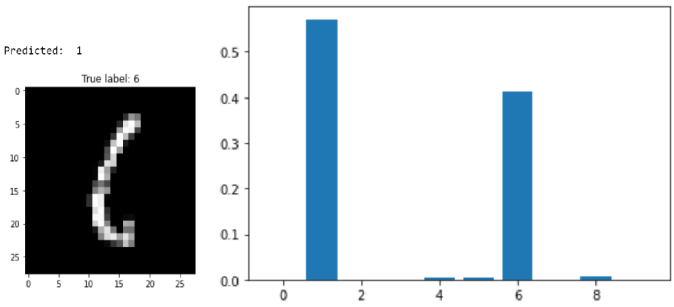
Although this takes a lot more time to train for MNIST data as compared to ANN, RNN gives better accuracy for test data at 99%.

Fig. 14. RNN accuracy



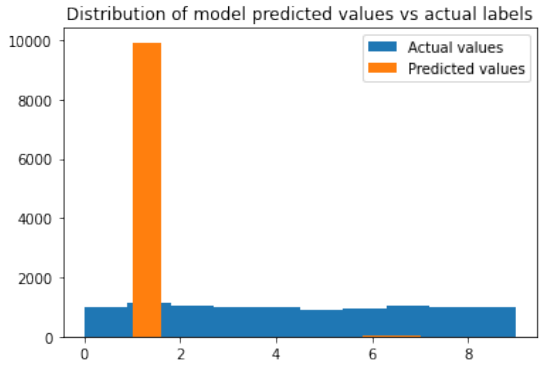
The mistakes made seem to be understandable. For eg, 6 in this case was labeled as 1. But we can see that the true label was a close second choice. However, its performance on

Fig. 15. RNN prediction error



noisy data is worse by labelling most data as 1. The accuracy is 11% which is essentially random. When faced with noise

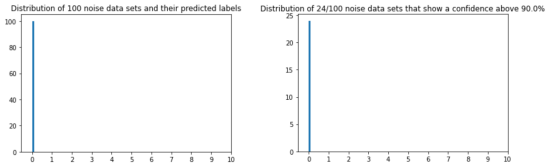
Fig. 16. RNN prediction on noisy data



data, LSTM gives out inconsistent results that are consistent in some ways. It sees a pure noise data as a single number with low, sometimes zero confidence. For example, in this case, all

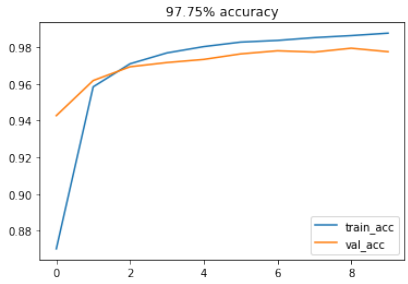
noise data is labelled as 0 with 24% of the datapoints predicted with high confidence. In another case, all noise data are views as 8 with zero datapoints predicted with high confidence. If

Fig. 17. RNN noise analysis



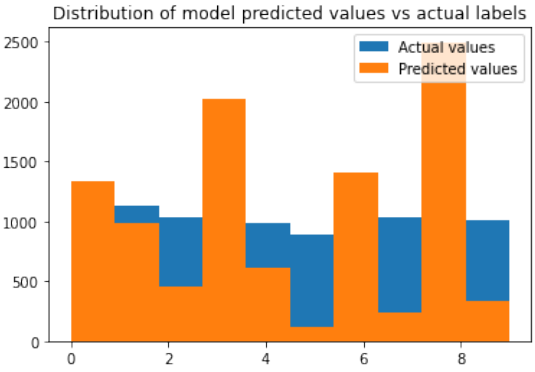
the RNN is Trained and validated using noisy data, it gives an accuracy of 98% higher than that of the ANN, however,

Fig. 18. RNN trained and validated on noisy data



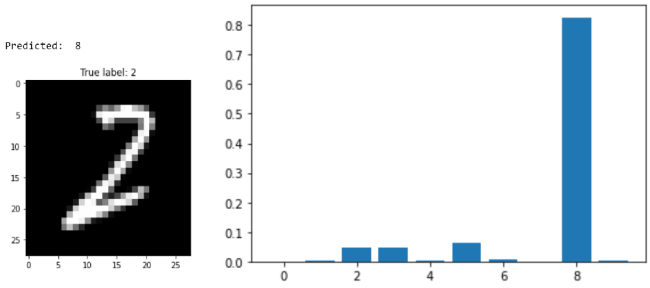
when this model is exposed to normal test data, it gives about 50% accuracy with the distribution as shown. In this case, the

Fig. 19. RNN prediction distribution



model was biased towards 3,6,8 while not labelling 5,7,9. This

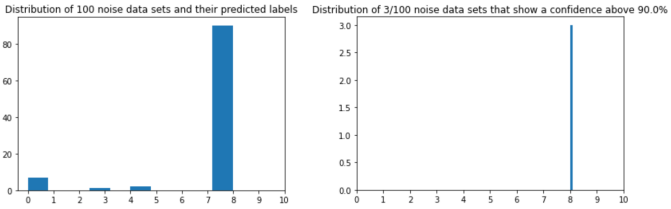
Fig. 20. RNN prediction error



might change but the bias towards 8 seems to be consistent. The mistakes made or not as human like as we saw previously. In the example shown in Fig 20, the number 2 which should seem obvious to anyone is not even in the top three guesses.

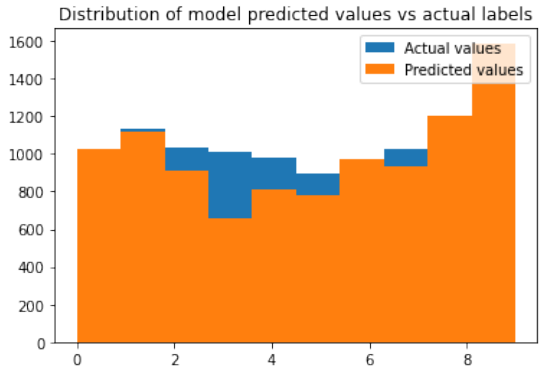
When it comes to noise, LSTM seem to recognize it much better with very few being highly confident in their predictions with most of them being 8, although that can change to other labels. When it comes to inverted images, the model

Fig. 21. RNN noise analysis



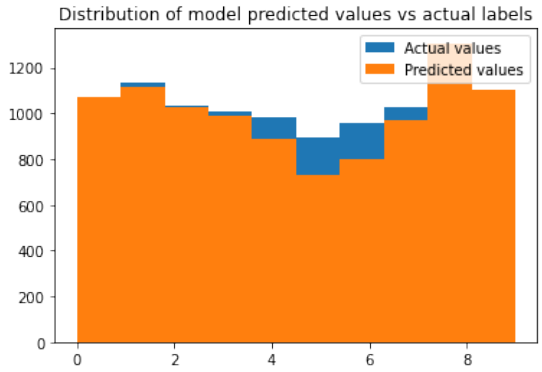
predictions are essentially random, usually a single number. The intelligence seems comparable to ANN by achieving 79% accuracy with 1000 training data. In this case, 3 and 4 are not

Fig. 22. prediction distribution



labelled as often by the model which is biased towards 9. If we try the threshold preprocessing as mentioned earlier where every value in the input matrix above 90% of the pixel value is changed to 100% while other pixels are put down to 0%, we get a very similar accuracy of 78% but a slightly different distribution. For some reason, the model does not label 4,5

Fig. 23. prediction distribution

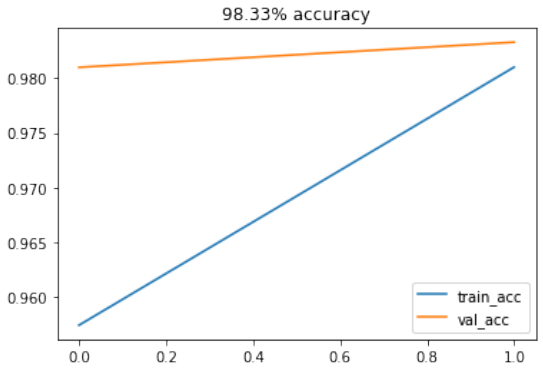


and 6 as often and is biased towards 8.

C. Convolutional Neural Network (CNN)

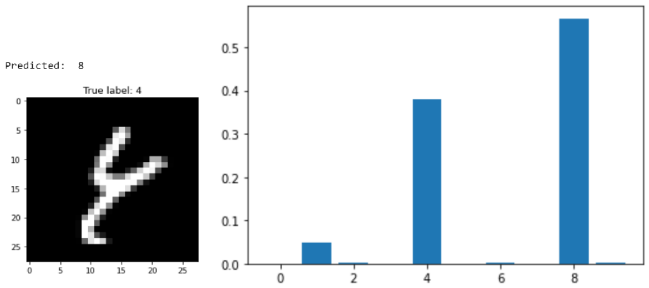
The CNN gives a 98% accuracy on standard MNIST testing data and it does so in just 2 epochs. However, unless it is done using a GPU, training this network takes time.

Fig. 24. CNN normal test accuracy



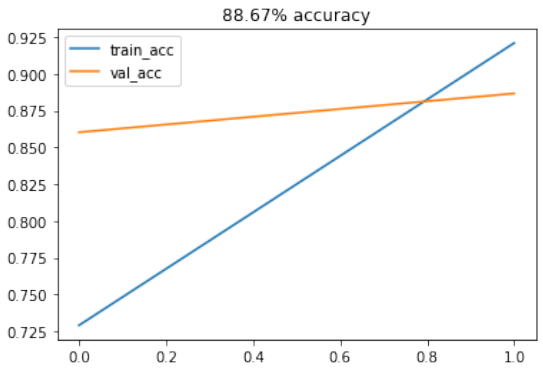
Oddly, mistakes made by CNN seem unintuitive from a human perspective, for instance, the digit below is predicted as 8, but its second guess is almost always correct.

Fig. 25. CNN prediction error



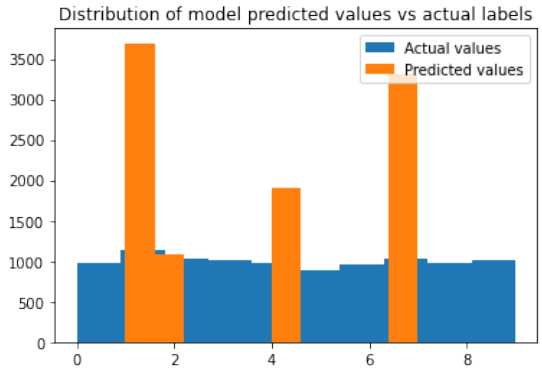
When we reduce training data to 1000, it gives an accuracy of 89%.

Fig. 26. CNN intelligence estimation



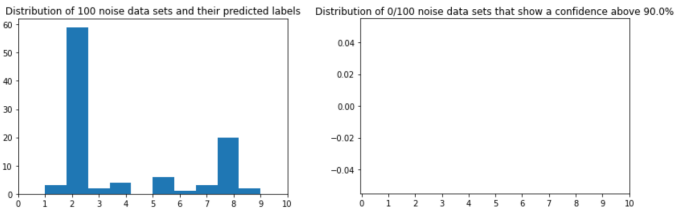
When this is faced with noisy data, the CNN gives predictions that vary from random to 20%. For some reason, it sees most noisy data as 1, and sometimes another number, like 7.

Fig. 27. CNN prediction trained on normal data, evaluated on noisy



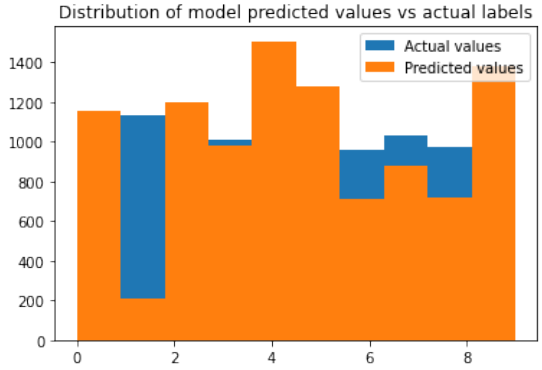
When it comes to noise, CNN views a lot of it as 8 and sometimes, another number like 2. However, it is not confident about its labels making it the most discerning model when it comes to noise.

Fig. 28. CNN Noise analysis



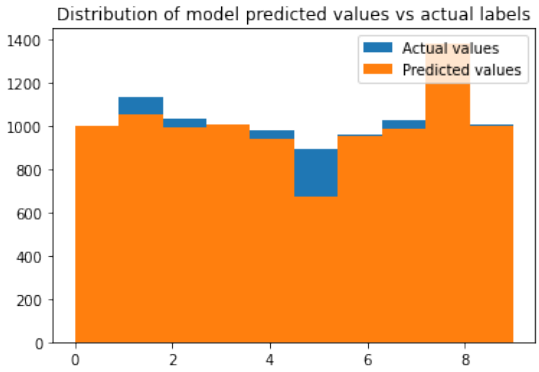
When faced with inverted data, CNN gives an impressive accuracy of 63% making it the best among all the models so far when it comes to image feature extraction. However, it fails to correctly label 1 while being biased towards 4 and 9 consistently.

Fig. 29. CNN prediction distribution on inverted data



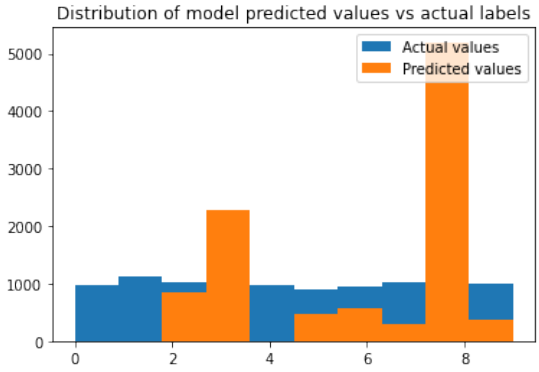
When the CNN is trained on noisy data, it gives an accuracy of 95% on noisy test data and 94% on normal test data. When it views normal data, it fails to see 5 and is biased towards 8 consistently.

Fig. 30. CNN trained on noisy, prediction distribution on normal data



When this is faced with inverted test data, it gives worse performance with an accuracy of 35%.

Fig. 31. CNN trained on noisy, prediction distribution on inverted data



When we look at the label distribution, it consistently views the inverted data as 8 and some other number like 3.

V. Conclusions

All three models gave good accuracy for standard MNIST data train tests. CNN gave high accuracy in just 2 epochs while for some reason, LSTM, an RNN tool to handle sequential data, gave the best test score for MNIST test images in this case. In cases where mistakes are made, most of the time it is very 'human' and the second guesses are usually the right answer. Gauging intelligence in terms of limited training data, RNN seems to be the loser with around 80% accuracy while ANN and CNN give results close to 90% with ANN often coming on top. When they are introduced to noisy data, ANN gives results of 20-50%, for RNN it is less than 15%, while for CNN it is less than 25%. When trained on noisy data, all three give good results for both normal and noisy data with CNN coming on top, ANN coming close behind and RNN having a hard time with normal MNIST data. Coming to noise analysis, ANN mislabelled too many noise data with very high confidence, with RNN being less sure and CNN showing the best performance by not having high confidence in labelling any noise data. After being trained with normal MNIST, RNN and ANN were unable to give good performance on inverted data, but CNN gave an adequate result of around 50-70% showing that it is the best among the lot for feature extraction.

All factors considered, CNN seems to give the best performance of them all but surprisingly, ANN comes in at a close second in so many key parameters. In fact, when it comes to intelligence in terms of limited training data, ANN seems to be a shade better. Then there is the fact it is also the most simplest, fastest and most intuitive among the three. Another observation is that in many cases, results seem to vary each time the models are created and trained showing the importance of initial weights. Perhaps we can one day examine their non-deterministic nature by setting weights to fixed values and see how it impacts model performance.

References

[1] David E. Rumelhart, Geoffrey E. Hinton Ronald J. Williams , "Learning representations by back-propagating errors ," Nature', 323, pages 533–536 1986.

[2] Hochreiter, Sepp; Schmidhuber, Jürgen (1997-11-01). "Long Short-Term Memory". Neural Computation. 9 (8): 1735–1780.

[3] Fukushima, Kunihiko (1980). "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position" (PDF). Biological Cybernetics.