Comparative study of Image Classifiers

R Sainiranjan
BS Mathenatics , Indian Institute Of Science
rsainiranjan@iisc.ac.in

Abstract

Image classification has historically and continues to occupy an important place in the field of Computer Vision . We have come a long way since the days when classifiers struggled to classify Handwritten digits and achieving human level accuracy seemed a distant pipedream . Modern classifiers have long since surpassed human level accuracy a fact substantiated by the results of the prestigious Image Net Competition where for several successive years modern Deep Learning architectures have won the contest with greater than 95 % accuracy causing the contest itself to be discontinued from 2017 ! The remarkable success of modern facial recognition systems is well known and in fact every day there seems to be some new approach or the other which manages to address problems that were hitherto unresolved and even seemed intractable ! This is an attempt to summarise some of the most commonly used architectures used in the field while showing their comparative advantages through results obtained on a particular set of data. We use Deep Learning based architectures they being a deep feed forward network , a simple convolutional neural network and modern convolutional based paradigms like AlexNet and ResNet .

1 Problem Statement

Image classification is an important field of Computer Vision. In classical image classification, a new class label is predicted and assigned to each piece of image data that summarises the main 'theme' in that image. Any image classification task mostly involves following tasks - data processing, features extraction, model building & evaluation. The data set used Fashion MNIST has 10 classes that can be used for classification - *'T-shirt/top', 'Trouser', 'Pullover', 'Dress','Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot'* that need to be classified. For this, efforts have been made to go slightly beyond the usual *"it depends"* answer to **"which model to use for classification?"**.There are many state of the art image classifiers which adopt various approaches to get appreciable results. This report is concentrated on classification models based on neural networks for classifying the above mentioned data set. It is *important to note* that constraints such as training and validation data have been kept constant for these models for a fair comparison of the models .

2 Data Description

There are 10 different data sets namely - *Tshirt/Top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag , Ankle Boot*. Each image is in grey-scale with an appropriate 'label' attached . The images are shown below

Figure 1: All data sets used

A brief description of the data:

There are in total 70000 such images available each has equal examples of each class . Machine Learning models need to be tested on unseen data to gauge their true on-field performance We have imported 60000 of these to train our model while the rest have been used as the test set. There has been no bias in selection and this has been done through tensor-flow's import data set functionality. Also, we need a subset of the data to be used as validation set in order to tune model parameters. So, This has been done by dividing the training set into thirds and using two thirds for the actual training and the last for validation purposes .

All our models have been implemented by using tensorflow 2.0 and the keras back-end that comes along with it . Link for the code has been provided at the end

3 Data Pre-Processing

To ensure that various models can be used the data has been engineered appropriately

3.1 Normalizationn

For the *Fashion MNIST* data set,each image is of the form 28*28 this is the matrix form representation of the pixelation of the image . Since values in the matrix can range from 0-255 it was necessary to normalise by dividing by 255 in order to ensure faster and more accurate fitting of the model.

3.2 Conversion of matrix to tensor

An important pre-processing step often glossed over in most academic work is the often necessary conversion of the image matrix into a tensor if the input has to be passed into a Convolutional Neural Network . The matrix was reshaped into the tensor (28,28,1) for our purposes

4 Modeling

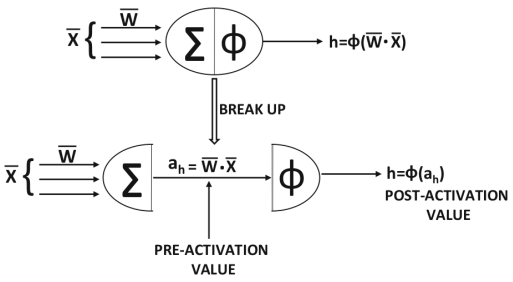
To better understand the modeling comparison and leanings mentioned in below sections, here are a few **assumptions that should be kept in mind**:

- While the models are non-standard in the implementations i.e while parameters such as nodes per layer and number of layers are not specific we have attempted to include as many to get a satisfactory result . Also we have tried our best to standardise these parameters across models to reduce their effects as con-founders towards reaching a conclusion.
- The intent of this exercise was to implement these models from scratch and understand the impact of how varying parameters effect the results
- While initially he goal was **not** to optimize the accuracy of the model but to understand why is the accuracy low and what can be done to boost it, these methods have naturally given rise to possible solutions which have been mentioned in a **Further Experiments** section.

4.1 Deep Feed Forward Network

In a single-layer network, a set of inputs is directly mapped to an output by using a generalized variation of a linear function. This simple instantiation of a neural network is also referred to as the perceptron. In multi-layer neural networks, the neurons are arranged in layered fashion, in which the input and output layers are separated by a group of hidden layers. This layer-wise architecture of the neural network is also referred to as a feed-forward network. As can be seen , in a neural

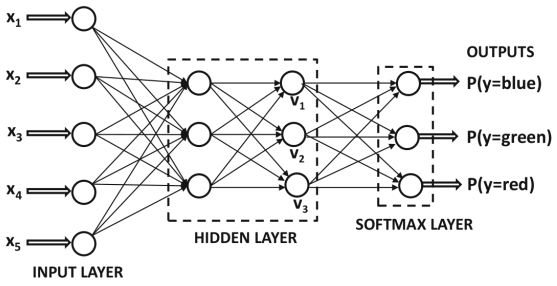
Figure 2: Working of a neural network



network first a linear combination of vectors of 'inputs' and 'weights' are taken which are passed through a non-linear */activation* function in order to 'learn' **non-linear** decision boundaries .

A multiple feed-forward network looks like what is shown in Figure 3 .

Figure 3: A Deep Feed Forward Neural network



Where the last softmax layer is necessary to output probabilities of each class .

The softmax function is defined as :

$$\Phi(\overline{v})_i = \frac{\exp(v_i)}{\sum_{j=1}^{k} \exp(v_j)} \quad \forall i \in \{1, \ldots, k\}$$

where the vector

$$\overline{v} = [v_1, \ldots, v_k]$$

represents the inputs to the softmax layer.

Our own feed forward neural network looks likes this :

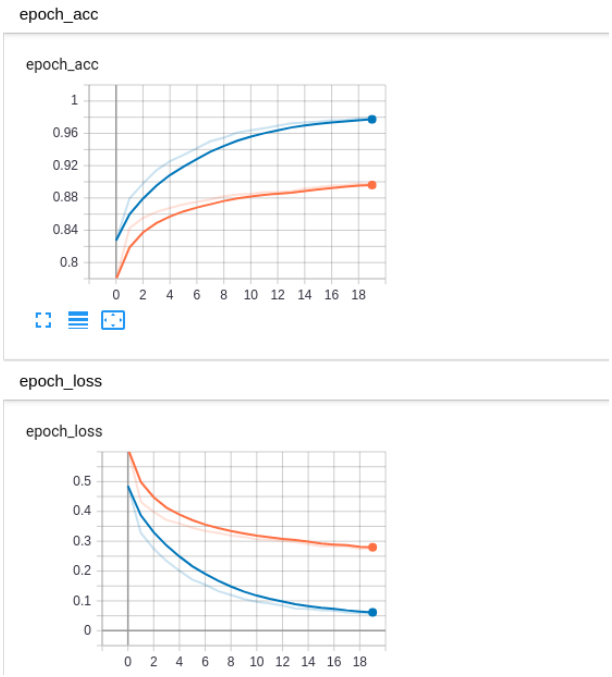
Figure 4: Deep Feed Forward Neural Network used for classification

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_5 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               100480
_________________________________________________________________
dense_4 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_5 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_6 (Dense)              (None, 10)                330
=================================================================
Total params: 111,146
Trainable params: 111,146
Non-trainable params: 0
```

The flatten layer is added to **flatten** our input . Recall that our neural network 'only' accepts vector and our image is a matrix and hence the flattening is necessary .The last Dense layer connects to a 10 way softmax layer for classification.

We show the results obtained after training here

Figure 5: Epoch loss and Epoch accuracy



These are epoch loss and epoch accuracy plots obtained from **tensorboard** with the y axes showing accuracy and loss with the validation set and the x axes showing the number of epochs used.

Our final evaluated accuracy came out to be 88.42 %

Figure 6: Test accuracy

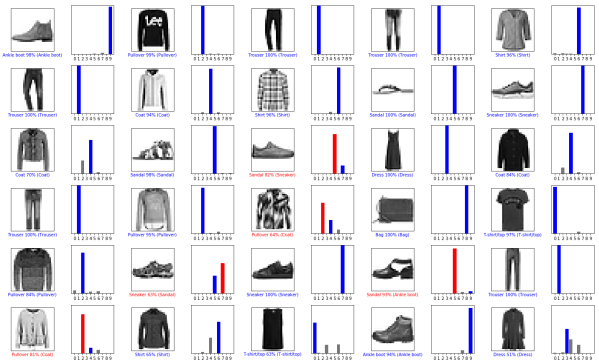
```
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('\nTest accuracy:', test_acc)              #test_Accuracy

10000/10000 [==============================] - 1s 110us/step

Test accuracy: 0.8842
```

To see a more understandable version of this result Recall that Fig 1 displayed a couple of sample images from the data-set now to show the results obtained after classification on the same samples

Figure 7: A visual display of our classifier



The figure clearly reveals how exactly our classifier works and more importantly fails on the test data . Clearly while we got a sufficient accuracy for our simplistic model part of the reason owes to the simplistic nature of our data-set as well . More complexity such as colours and problems such as occlusion are enough to confound the best of the feed forward neural networks which motivates us for the next model in our list

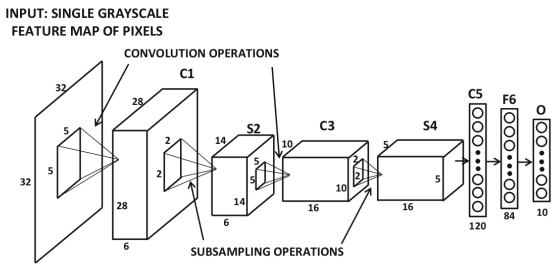
4.2 Convolutional Neural Network

Convolutional neural networks are biologically inspired networks that are used in computer vision for image classification and object detection. The basic motivation for the convolutional neural network was obtained from Hubel and Wiesel's understanding of the workings of the cat's visual cortex, in which specific portions of the visual field seemed to excite particular neurons. This broader principle was used to design a sparse architecture for convolutional neural networks. The first basic architecture based on this biological inspiration was the neocognitron, which was then generalized to the LeNet-5 architecture . In the convolutional neural network architecture, each layer of the network is 3-dimensional, which has a spatial extent and a depth corresponding to the number of features. The notion of depth of a single layer in a convolutional neural network is distinct from the notion of depth in terms of the number of layers. In the input layer, these features correspond to the color channels like RGB (i.e., red, green, blue), and in the hidden channels these features represent hidden feature maps that encode various types of shapes in the image. If the input is in grayscale (like LeNet-5), then the input layer will have a depth of 1, but later layers will still be 3-dimensional. The architecture contains two types of layers, referred to as the convolution and subsampling layers, respectively.

For the convolution layers, a convolution operation is defined, in which a filter is used to map the activations from one layer to the next. A convolution operation uses a 3-dimensional filter of weights with the same depth as the current layer but with a smaller spatial extent. The dot product between all the weights in the filter and any choice of spatial region (of the same size as the filter) in a layer defines the value of the hidden state in the next layer (after applying an activation function like ReLU). The operation between the filter and the spatial regions in a layer is performed at every possible position in order to define the next layer (in which the activations retain their spatial relationships from the previous layer). The connections in a convolutional neural network are very sparse, because any activation in a particular layer is a function of only a small spatial region in the previous layer.

All layers other than the final set of two of three layers maintain their spatial structure. Therefore, it is possible to spatially visualize what parts of the image affect particular portions of the activations in a layer. The features in lower-level layers capture lines or other primitive shapes, whereas the features in higher-level layers capture more complex shapes like loops (which commonly occur in many digits). Therefore, later layers can create digits by composing the shapes in these intuitive features. This is a classical example of the way in which semantic insights about specific data domains are used to design clever architectures.

Figure 8: A sample Convolution Neural Network



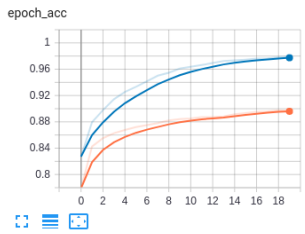
The above figure shows a sample convolutional neural network. Our own convolution neural network looks like this :

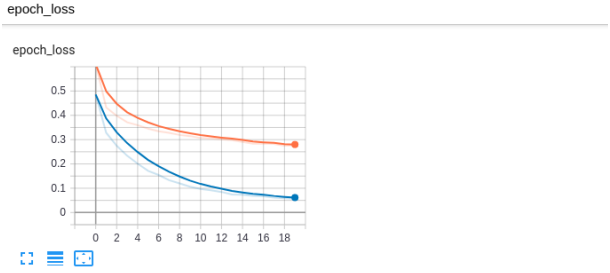
Figure 9: The implemented Convolution Neural Network

```
Model: "sequential_16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_414 (Conv2D)          (None, 28, 28, 64)        320
_________________________________________________________________
max_pooling2d_41 (MaxPooling (None, 28, 28, 64)        0
_________________________________________________________________
dropout_43 (Dropout)         (None, 28, 28, 64)        0
_________________________________________________________________
conv2d_415 (Conv2D)          (None, 28, 28, 32)        8224
_________________________________________________________________
max_pooling2d_42 (MaxPooling (None, 28, 28, 32)        0
_________________________________________________________________
dropout_44 (Dropout)         (None, 28, 28, 32)        0
_________________________________________________________________
flatten_22 (Flatten)         (None, 25088)             0
_________________________________________________________________
dense_51 (Dense)             (None, 256)               6422784
_________________________________________________________________
dropout_45 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_52 (Dense)             (None, 10)                2570
=================================================================
Total params: 6,433,898
Trainable params: 6,433,898
Non-trainable params: 0
```

Dropout layers are added to introduce 'regularisation' where we deliberately randomly choose neurons to 'forget' connections to prevent overfitting . We perform sampling by max-pooling (pass a filter and perform the convolution operation choose the maximum element in the final matrix) . Flatten figures here as well as in the end we need to pass our transformed input as a vector to get results . Here's how some of the results look like

Figure 10: Epoch loss and accuracy





Our final accuracy came out to be **91.26%**

Figure 11: Test accuracy

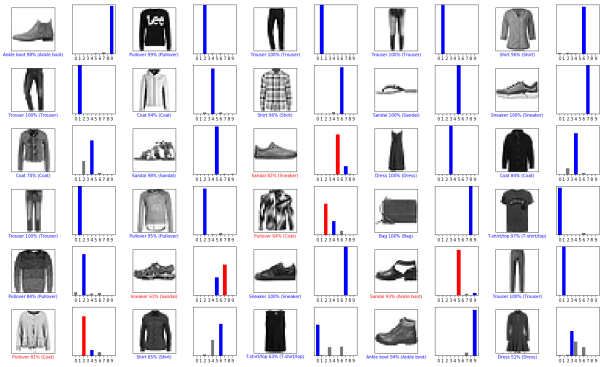
```
test_loss, test_acc = model2.evaluate(test_images3, test_labels)
print('\nTest accuracy:', test_acc)

10000/10000 [==============================] - 1s 142us/sample - loss: 0.3939 - acc: 0.9126

Test accuracy: 0.9126
```

Once again a visualisation for more clarity

Figure 12: A visual display of our classifier



Notice that when compared to our previous plot that in objects that are incorrectly classified by both classifiers the percentage of surety decreases in this network which means that the model is 'less sure' that it is making a correct decision along with the higher accuracy this tells us that this model is 'learning ' to perform better than the previous one!

4.3 AlexNet

AlexNet is the name of a convolutional neural network (CNN), designed by Alex Krizhevsky, and published with Ilya Sutskever and Krizhevsky's doctoral advisor Geoffrey Hinton.

AlexNet competed in the ImageNet Large Scale Visual Recognition Challenge on September 30, 2012. The network achieved a top-5 error of 15.3 %, more than 10.8 percentage points lower than that of the runner up. The original paper's primary result was that the depth of the model was essential for its high performance, which was computationally expensive, but made feasible due to the utilization of graphics processing units (GPUs) during training .

AlexNet is a 8 layer convolutional neural network architecture which consists of Convolutional layers, Activation layers, Max Pooling layers and Dense layers. It has 5 convolutional layers, 3 max pooling layers, and 3 dense layers with one output dense-softmax layer.

Conv1 consists of 11x11 filters, while Conv2 uses 5x5 filters, and Conv3 uses 3x3 filters, Conv4 uses 3x3 filters and Conv5 uses 3x3 filters.Here we show our implementation of AlexNet

Figure 13: AlexNet implemented by us

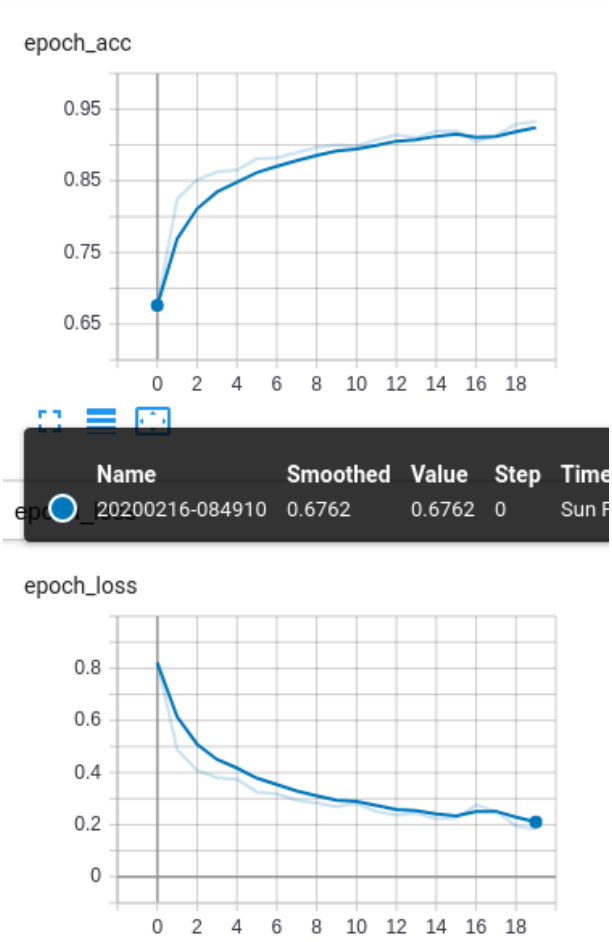
Layer (type)	Output Shape	Param #
conv2d_378 (Conv2D)	(None, 5, 5, 96)	11712
max_pooling2d_19 (MaxPooling	(None, 5, 5, 96)	0
conv2d_379 (Conv2D)	(None, 5, 5, 256)	614656
max_pooling2d_20 (MaxPooling	(None, 5, 5, 256)	0
conv2d_380 (Conv2D)	(None, 5, 5, 384)	885120
dropout_1 (Dropout)	(None, 5, 5, 384)	0
conv2d_381 (Conv2D)	(None, 5, 5, 384)	1327488
conv2d_382 (Conv2D)	(None, 5, 5, 256)	884992
max_pooling2d_21 (MaxPooling	(None, 5, 5, 256)	0
dropout_2 (Dropout)	(None, 5, 5, 256)	0
flatten_8 (Flatten)	(None, 6400)	0
dense_14 (Dense)	(None, 9216)	58991616
dense_15 (Dense)	(None, 4096)	37752832
dense_16 (Dense)	(None, 4096)	16781312
dropout_3 (Dropout)	(None, 4096)	0
dense_17 (Dense)	(None, 10)	40970

Total params: 117,290,698
Trainable params: 117,290,698
Non-trainable params: 0

It is not much different from our earlier implementation and in fact our earlier implementation was in some sense a slightly simplified version of this network.

Once again we present our results here :

Figure 14: Epoch loss and accuracy



The test accuracy this time was **92.93** % which is a marginal increase from the earlier model.

We do not present the visualisation for this model as it is highly identical to the previous one with very subtle differences and hence it isn't that useful for inferences.

An interesting problem we noted was the rapid loss in accuracy in both the validation and test sets as we added more and more convolutional layers . Consider this 'Deep AlexNet' for example with 11 layers

Figure 15: Deeper AlexNet

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_43 (Conv2D)           (None, 5, 5, 96)          11712
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 5, 5, 96)          0
_________________________________________________________________
conv2d_44 (Conv2D)           (None, 5, 5, 256)         614656
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 5, 5, 256)         0
_________________________________________________________________
conv2d_45 (Conv2D)           (None, 5, 5, 384)         885120
_________________________________________________________________
conv2d_46 (Conv2D)           (None, 5, 5, 384)         1327488
_________________________________________________________________
dropout_19 (Dropout)         (None, 5, 5, 384)         0
_________________________________________________________________
conv2d_47 (Conv2D)           (None, 5, 5, 256)         884992
_________________________________________________________________
conv2d_48 (Conv2D)           (None, 5, 5, 256)         590080
_________________________________________________________________
conv2d_49 (Conv2D)           (None, 5, 5, 256)         590080
_________________________________________________________________
dropout_20 (Dropout)         (None, 5, 5, 256)         0
_________________________________________________________________
conv2d_50 (Conv2D)           (None, 5, 5, 256)         590080
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 5, 5, 256)         0
_________________________________________________________________
flatten_8 (Flatten)          (None, 6400)              0
_________________________________________________________________
dropout_21 (Dropout)         (None, 6400)              0
_________________________________________________________________
dense_29 (Dense)             (None, 9216)              58991616
_________________________________________________________________
dense_30 (Dense)             (None, 4096)              37752832
_________________________________________________________________
dense_31 (Dense)             (None, 4096)              16781312
_________________________________________________________________
dense_32 (Dense)             (None, 10)                40970
=================================================================
Total params: 119,060,938
Trainable params: 119,060,938
Non-trainable params: 0
```

When we trained this model we obtained the result as shown

Figure 16: Test Accuracy of Deep Alex Net

```
predictions4 = model3.predict(test_images3)
test_loss, test_acc = model3.evaluate(test_images3, test_labels)
print('\nTest accuracy:', test_acc)
```

```
10000/10000 [==============================] - 2s 162us/step

Test accuracy: 0.1
```

What an astonishing drop in accuracy ! From 90% to 10% !
Intuitively deeper layers should work better however in practice it often doesn't work out due to issues such **Vanishing** or **Exploding** Gradients or 'dying ' out of activation functions are problems in itself but at least in our Alex Net case we can ignore these issues (We are in effect adding three extra layers and if the problem was due to these causes the drop in accuracy would have been less gradual !)

In that case why then does training deeper layers lead to more layers ? Quite counter-intuitive isn't it ? Yet this issue is encountered more often than we like and needs to be taken care of if we wish to build and deploy more robust models .
The phenomenon encountered is termed in academia as **Degradation** which prevents the creation of deeper layers and thus restricts the power of our models . We shall use this problem as a motivation to construct our next and final model the ResNet

4.4 ResNet

The problem of **Degradation** discussed above tantamounts to the difficulty in getting the learning process to converge properly in a reasonable amount of time. Such convergence problems are common in networks with complex loss surfaces.

Although some deep networks show large gaps between training and test error, the error on both the training ad test data is high in many deep networks. This implies that the optimization process has not made sufficient progress .

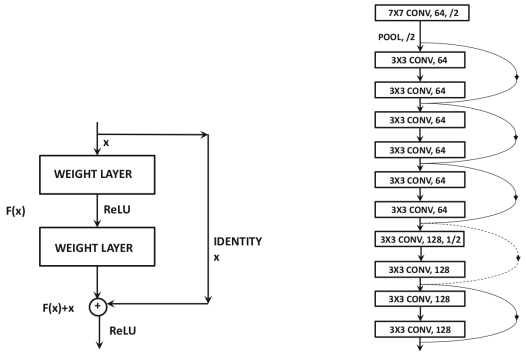
This problem is exacerbated by deeper networks ; Since all features do not have the same level of abstraction i.e some concepts can be learned by using shallow networks, whereas others require fine-grained connections .

For eg. consider a circus elephant standing on a square frame. Some of the intricate features of the elephant might require a large number of layers to engineer, whereas the features of the square frame might require very few layers. Convergence will be unnecessarily slow when one is using a very deep network with a fixed depth across all paths to learn concepts, many of which can also be learned using shallow architectures.

A natural solution that arises is to let the neural network itself **decide** how many layers to use to learn each feature .

ResNet uses skip connections between layers in order to enable copying between layers and introduces an iterative view of feature engineering (as opposed to a hierarchical view) . Most feed-forward networks only contain connections between layers i and (i + 1), whereas ResNet contains connections between layers i and (i + r) for r $>$ 1. An example has been shown in the figure below.

Figure 17: A sample ResNet



This skip connection simply copies the input of layer i and adds it to the output of layer (i + r). Such an approach enables effective gradient flow because the back-propagation algorithm now has a super-highway for propagating the gradients backwards using the skip connections. This basic unit is referred to as a residual module, and the entire network is created by putting together many of these basic modules.

In most layers, an appropriately padded filter 7 is used with a stride of 1, so that the spatial size and depth of the input does not change from layer to layer. In such cases, it is easy to simply add the input of the ith layer to that of (i + r). However, some layers do use strided convolutions to reduce each spatial dimension by a factor of 2. At the same time, depth is increased by a factor of 2 by using a larger number of filters. In such a case, one cannot use the identity function over the skip connection. Therefore, a linear projection matrix might need to be applied over the skip connection in order to adjust the dimensionality. This projection matrix defines a set of 1 1 convolution operations with stride of 2 in order to reduce spatial extent by factor of 2. The parameters of the projection matrix need to be learned during backpropagation.

In the original idea of ResNet, one only adds connections between layers i and (i+r). For example, if we use r = 2, only skip connections only between successive odd layers are used. Later enhancements like DenseNet showed improved performance by adding connections between all pairs of layers.

Our model looks like in Figure 18

This is quite a deep model and implementing it would have been not possible if the residual mappings (identity activations in our case) weren't included .

The results we obtained were as shown in Figures 19 and 20

Notice that the accuracy is back again to 90.76 % ! This is the power of this model which leverages *Residual* connections to over-come degradation.

Once again a visualization for more clarity is presented in Figure 21.

ResNet and the modern architectures based on ResNet have been at the forefront of research and deployment in the field of Computer Vision i.e they are in some sense the 'cutting-edge' models of the field being able to leverage the abundant computational resources that are at the fingertips of most organizations in order to make better insights than that had been possible before.

Figure 18: Our ResNet

Figure 19: Epoch loss and accuracy

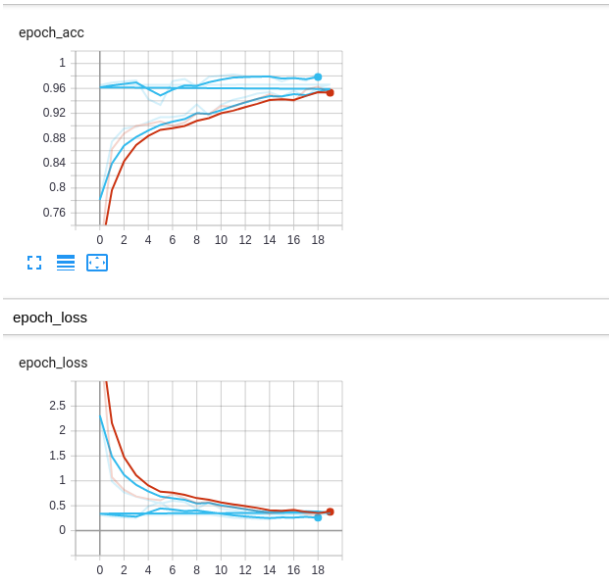


Figure 20: Test accuracy

```
predictions3 = res_net_model.predict(test_images3)
test_loss, test_acc = res_net_model.evaluate(test_images3, test_labels)
print('\nTest accuracy:', test_acc)
```

```
10000/10000 [==============================] - 3s 301us/step

Test accuracy: 0.9076
```

Figure 21: Visualization Of Results

5 Further Experiments

Now , that we have discussed the basic working of several architectures and have shown some results it makes sense at this point to discuss possible investigations that can lead to greater insights and hopefully spawn some research that can eventually lead to breakthroughs !

5.1 Data Augmentation

Data Augmentation refers to the deliberate addition of noise to the data-set in order to force the model to generalise better . We on different instances performed random rescaling, horizontal and vertical flips, perturbations to brightness, contrast, and color, as well as random cropping. While the flips increased the accuracy moderately

we found out that random rescaling performed better as an augmentation technique . However when cropping was combined with patching the test accuracy worked out to be even better . With more and more methods available for data augmentation and regularization in general the development of more tailored methods depending on the dataset is an issue certainly worth an investigation.

5.2 Fine-tuning Parameters (Behind the Hood)

This involved adjusting for the validation size, activation functions, filter size , number of nodes per layer etc.
We observed that the validation set of size between 0.20 and 0.35 provided the best results as higher sizes led to **underfitting** and lower sizes often showed a reduced extent of generalisation .
Among the vast amount of ready-made activation functions available to use such as tanh , sigmoid ,relu etc . Relu gave us the best results unfortunately it is increasingly unreliable in networks with greater depth and thus prompts adoption of methods such as Leaky ReLu which can ensure that the function does not collapse even if many successive gradients pass through .
We fixed the size of filters to be typically in powers of 2 since they support easy reduction (just divide by 2 !) and easy scaling (just multiply by 2 !) Though other sizes have also been used to test for any effects . Pooling operations are typically 9implemented as MaxPooling however Average Pooling were also used once or twice again this depends on the deployer and the nature of the dataset. Node sizes of the feed forward layers were also taken in the powers of 2 to give some sense of continuity from the convolutional layers . Optimizer chosen has always been the Adam optimizer which performs far better than the conventional Stochastic Gradient Descent optimizer . Loss function has been Sparse Categorical Cross-Entropy to ensure that we get classification into discrete classes.

So far the task of fine-tuning these parameters has been more of an art than a science and while the consequences are not as apparent when simple feed forward neural networks are used in more complex models consider the ResNet for example; variation in number of layers , number of nodes and the filter size can lead to drastically different results and hence these models often require hours of careful calibration in order to get what can pass for an acceptable result . With thresholds for acceptability being non-standard it is understandable while it is still preferable in commercial settings to deploy less complex models which although may not have as high an accuracy but are not as **brittle** (dependent on the parameters) as these complex models. A still open problem is to figure out efficient means of computing these parameters so that we can deploy these systems efficiently.

5.3 Generative Modelling

A natural means of introducing better generalisation is naturally to *'Expand'* the Dataset . But how do we do this when we are in fact not the authors of the actual dataset ? We can leverage the power of **Generative Modelling** to do this . We use GANs (Generative Adversarial Networks) a Generative Model that learns the underlying *Probability Distribution* of the images and eventually learns how to generate an image of it's own when random noise is provided as an input !
Generative adversarial networks work with two neural network models simultaneously. The first is a generative model that produces synthetic examples of objects that are similar to a real repository of examples. Furthermore, the goal is to create synthetic objects that are so realistic that it is impossible for a trained observer to distinguish whether a particular object belongs to the original data set, or whether it was generated synthetically. For example, if we have a repository of car images, the generative network will use the generative model to create synthetic examples of car images. As a result, we will now end up with both real and fake examples of car images. The second network is a discriminative network that has been trained on a data set which is labeled with the fact of whether the images are synthetic or fake. The discriminative model takes in inputs of either real examples from the base data or synthetic objects created by the generator network, and tries to discern as to whether the objects are real or fake. In a sense, one can view the generative network as a "counterfeiter" trying to produce fake notes, and the discriminative network as the "police" who is trying to catch the counterfeiter producing fake notes. Therefore, the two networks are adversaries, and training makes both adversaries better, until an equilibrium is reached between them.

The images generated by the GAN are shown here

Figure 22: Generated by a GAN

Clearly these images are virtually identical to those from our dataset . For better results we used a conditional GAN since that can based on the constraint that each image belongs to a certain class learn faster and more accurately than traditional vanilla GANs . There are more interesting architectures of GANs available like Style GANs , Cycle GANs etc that are capble of far more powerful transformations . To leverage the data generated by GANs however we needed to create implicit labels in order to ensure that they could be incorporated into our 'training set' . After this procedure we observed a heightened accuracy as what we are performing amounts to a much more sophisticated means of data augmentation as the images generated although similar are not exactly identical to those present in the dataset.

Further investigations along this approach can no doubt be constructed and in fact one of the main usages of Generative Modelling (unlike the public perception of generating fake images !) is to generate more examples that can lead to better generalisation of models trained on datasets . Adapting GANs to areas like textual classification and improving GANs for image classification purposes is an active area of research.

6 Conclusion

We have tracked the progress of Deep Neural Network based Image Classifiers through our dataset Fashion MNIST (This dataset may not be able to show drastic differences between models but that's exactly why it is essential since almost all models work and their results can be tracked !) We have tried our best to provide a logical progress upward in the direction of the ladder of Image Classification through the means of a comparative analysis . By no means is this list exhaustive or up to date methods like Bayesian Modelling which can provide excellent inference haven't been discussed here for example and incessant innovations in the field of Computer Vision might bring up far more better models than the ones used here. Still as a starting point , we believe that our exposition fulfils its purpose.

References

[1] Ian Goodfellow , Yoshua Bengio and Aaron Courville *Deep Learning*. MIT Press , Massachusetts , 2016

[2] Ryo Takahashi, Takashi Matsubara and Kuniaki Uehara *Data Augmentation using Random Image Cropping and Patching for Deep CNNs* JOURNAL OF LATEX CLASS FILES, VOL. 14, NO. 8, AUGUST 2015 1