ELL 793
Assignment 2

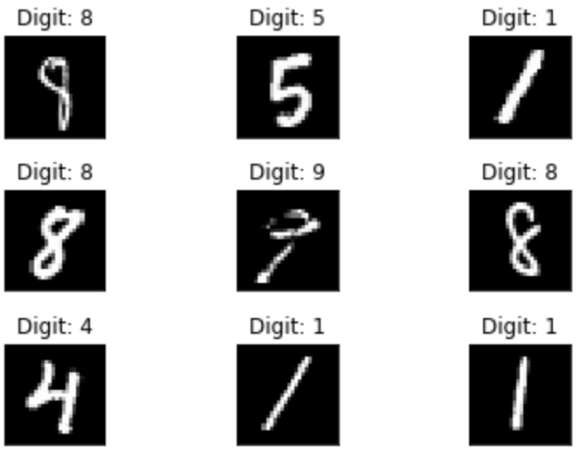
24th Nov 2021

Submitted by:

Sarthak Garg (2017EE30546)
Varun Gupta (2017EE30551)

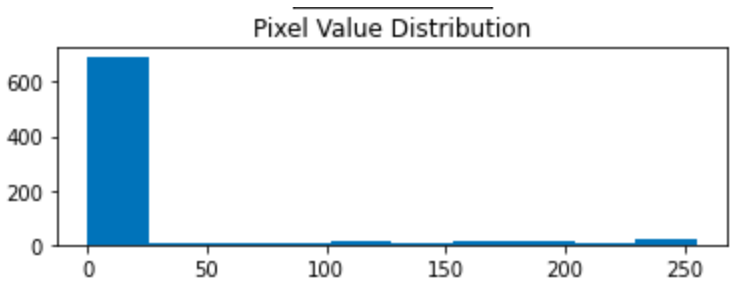
PROBLEM 1: Classification on MNIST dataset

Visualizing random images from the MNIST Dataset



Statistical analysis on the content of the feature vector

We do this by seeing the pixel value distribution for all the images in the dataset



As expected, most of the input features are 0 since the image backgrounds are black (0) and the text is written with white (255). More than 600 pixels on average have a value of 0 or very close to 0.

Data Preprocessing

1. The loaded data has been split into train, test, and cross_val datasets
2. The cross-validation dataset has been chosen by picking by random shuffling of training set
3. **One-hot encoding** has been done on the y data so that for each data input, the y value is of size 10 with 1 at the index of the correct digit. For example, the y value corresponding to the digit 3 will be all 0s and 1 in the 3rd index.
4. The datasets have been reshaped as follows:

```
X_train shape (50000, 784)
y_train shape (50000, 10)
X_val shape (10000, 784)
y_val shape (10000, 10)
X_test shape (10000, 784)
y_test shape (10000, 10)
```

5. **Normalization:** It has been widely accepted that image datasets can be normalized by dividing all the pixel values by the max pixel value. Here we have divided all the NumPy matrices by 255 to normalize them between 0 and 1.

Model Used

We have used the following sequential model

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_9 (Dense)              (None, 500)               392500
_________________________________________________________________
activation_9 (Activation)    (None, 500)               0
_________________________________________________________________
dense_10 (Dense)             (None, 500)               250500
_________________________________________________________________
activation_10 (Activation)   (None, 500)               0
_________________________________________________________________
dense_11 (Dense)             (None, 10)                5010
_________________________________________________________________
activation_11 (Activation)   (None, 10)                0
=================================================================
Total params: 648,010
Trainable params: 648,010
Non-trainable params: 0
```

We have compiled the model using the following parameters:

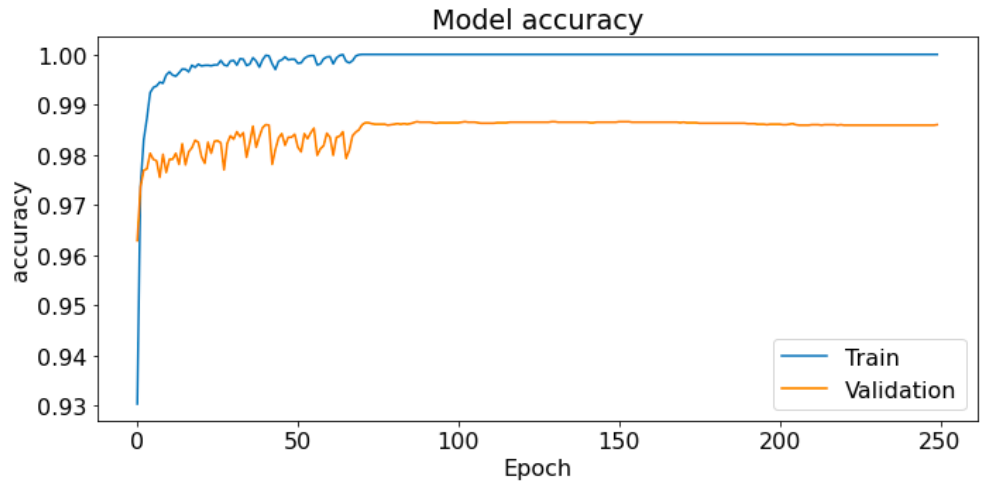
1) Loss function: Categorical cross-entropy
2) Metric: Accuracy
3) Optimizer: Adam

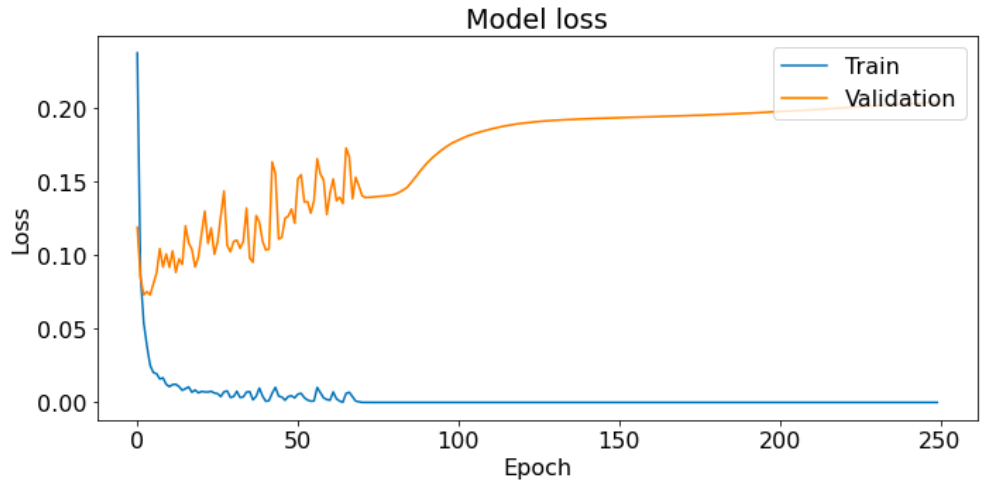
Why do we choose this loss function?

The categorical cross-entropy loss function is usually used in single labeled multiclass classification because this loss is a very good measure of how distinguishable two discrete probability distributions are from each other. This works very well because we have a set of 10 images and basically need to distinguish between them.

Results

Classification Accuracy and Error on training and validation data





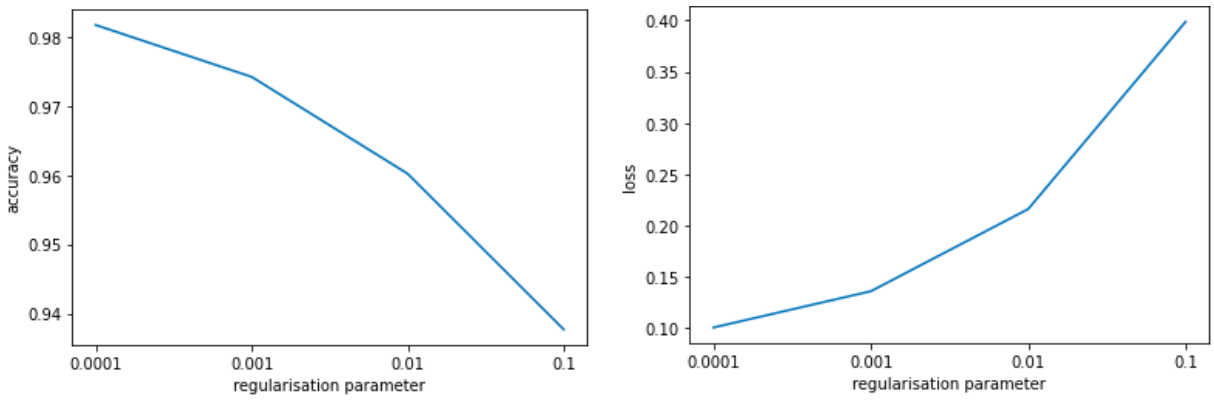
The test accuracy is **0.982** and the loss is **0.1313**
We can infer the following results from the plots above:

1) The model always performs better on train data than cross-validation data.
2) After around 20- 30 epochs, the training accuracy and the validation become almost constant indicating that no real training is happening after this.
3) After a certain number of epochs, the loss on cross-validation starts increasing which indicates that the model is overfitting on the training set. Hence, it is not performing well on the cross validation set
4) Almost 80% of the epochs are not useful and we can use some early stopping conditions to save time and avoid overfitting.

Implementing L2 Regularisation

l2 regularisation was implemented using keras regulariser. The value of regularisation parameter lambda is varied in `['0.0001', '0.001', '0.01', '0.1']`.

The accuracy and loss is compared on validation data, after training on 100 epochs, and the plots are as shown below.

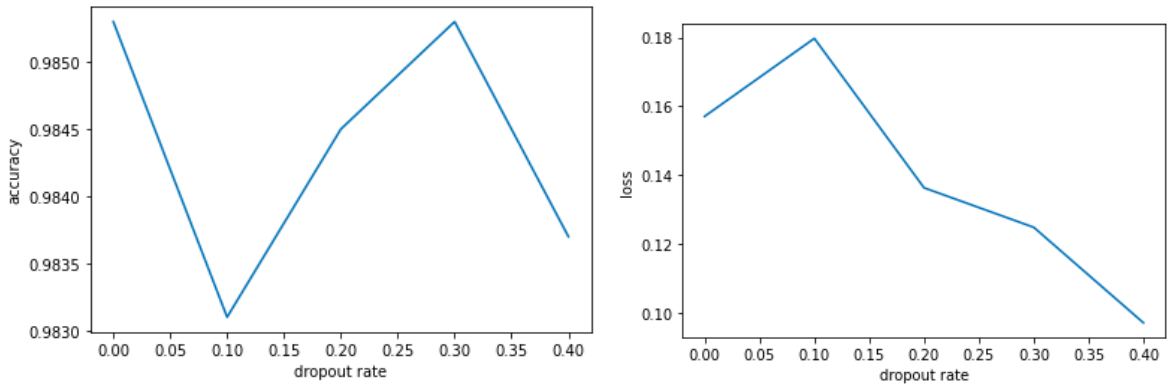


The maximum accuracy (`0.9817`) and lowest loss(`0.100`) is obtained at the lowest value of regularization parameter = 1e-4. Apparently, increasing regularization above this is causing the model to lose complexity and so the accuracy is falling significantly.

Implementing Dropout

Dropout was implemented using the keras layer `model.add(Dropout(rate))`, where rate is the fraction of units to drop

Dropout rate was varied from 0 to 0.3 in steps of 0.1 and the validation accuracy and loss was compared after training on 100 epochs. The plots of validation accuracy and loss are shown below.

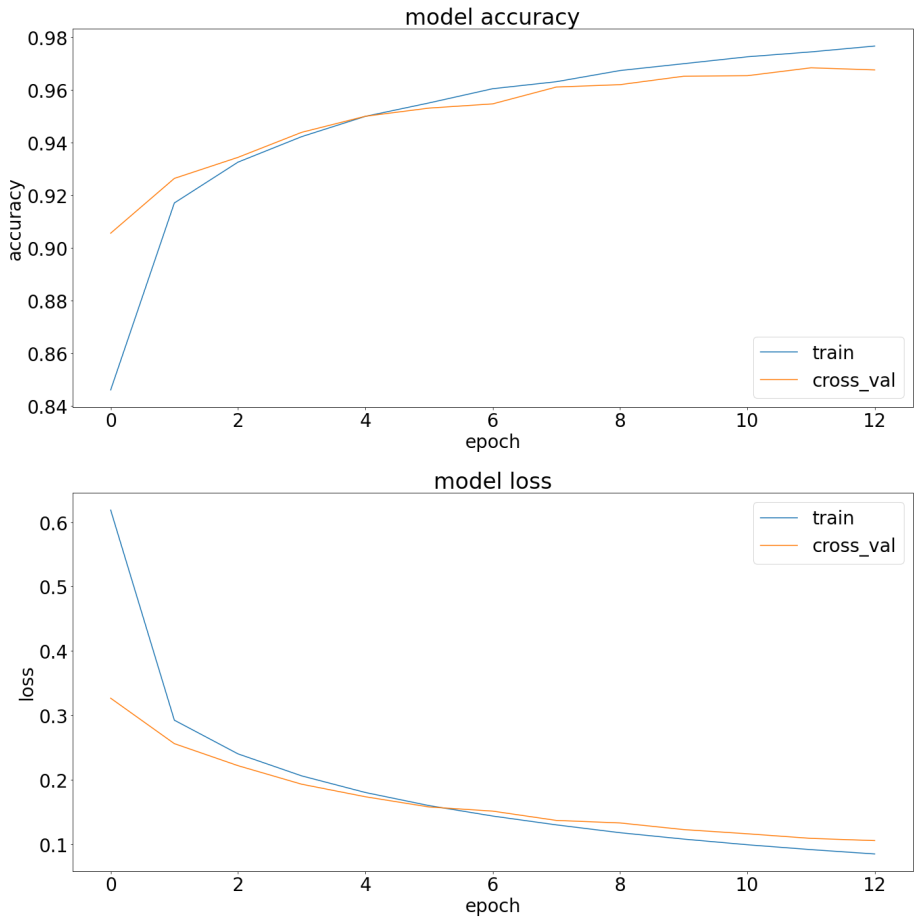


Best possible combination of high accuracy(`0.9853`) and low loss(`0.1247`) is at dropout rate = 0.3. This is expected because as seen before, this model severely overfits after 50 epochs and

so the regularisation should be significant to prevent this overfitting and so higher rate of dropping inputs is desired to make the model more simple so that the overfitting is reduced.

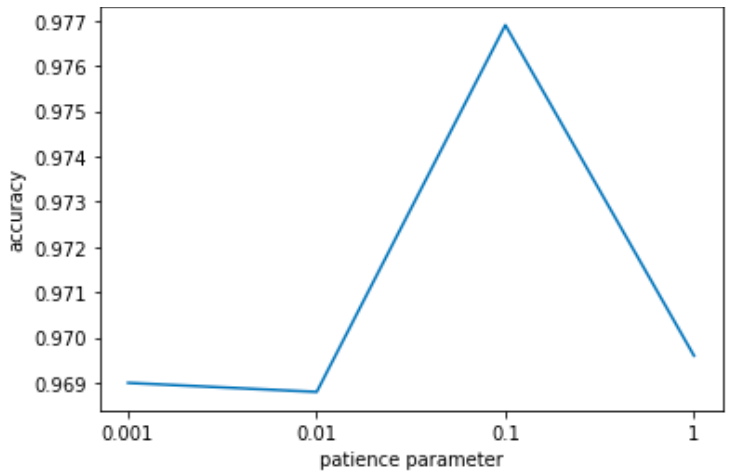
Implementing early stopping

Early stopping was implemented using the keras layer `es = EarlyStopping(monitor='val_acc', mode='auto', verbose=1, patience=p)`

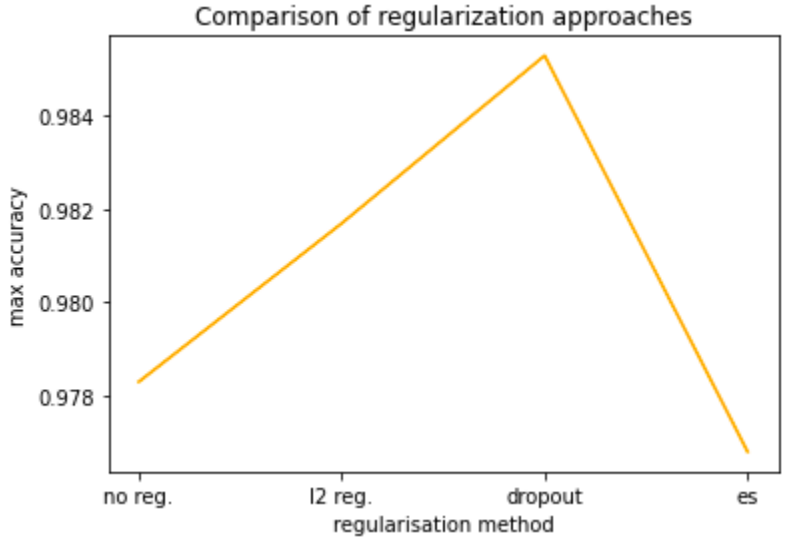


From the above plot we can infer that the early stopping condition stopped the training at **12** epoch and we achieved a validation accuracy of **0.9768.**

Next, We varied the patience parameter and achieved highest validation accuracy at **patience=0.1**.



Comparison of regularisation methods

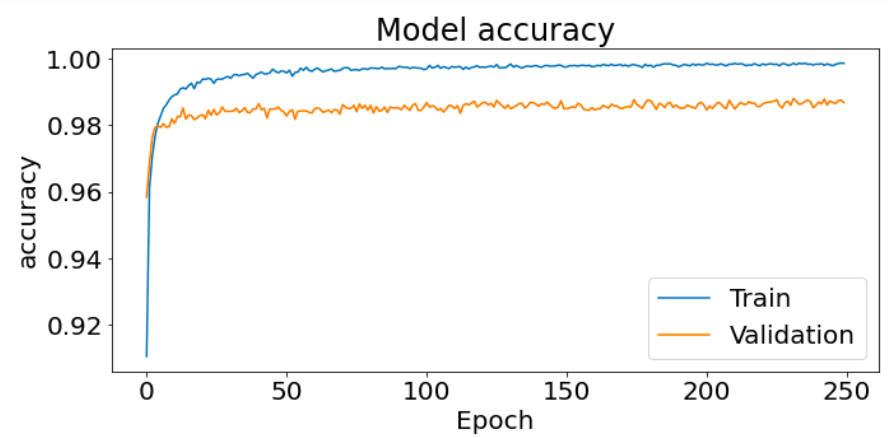


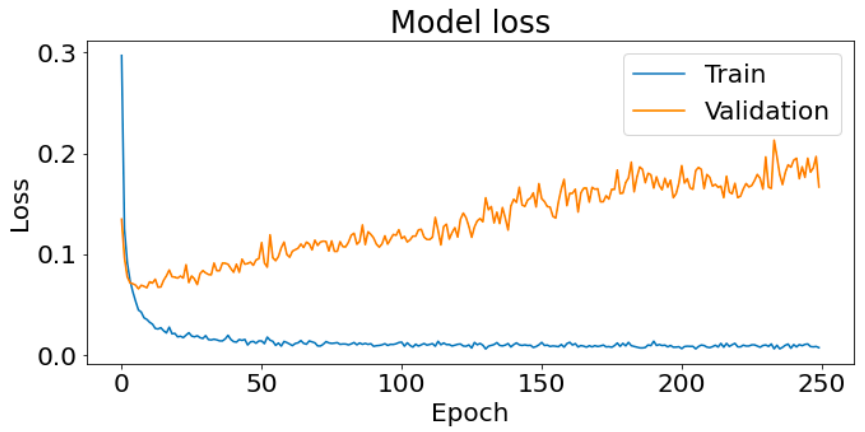
It turns out that for our selection of model parameters and training on 100 epochs, dropout is performing the best among the regularisation methods and is performing better than without applying regularization.

This can be intuitively understood as well. Dropout works well for complex networks to introduce linearity in the otherwise non-linear overfitted function that the neural network has learned. Here, MNIST is a relatively simple dataset and we are also using 500 neurons in the hidden layers which is adding to a lot of complexity. Dropout is dropping some of these intermediate inputs during training which reduces the complexity of the function and hence is performing well, even when the number of epochs are large.

Final Model after applying Regularization

Finally, we ran the entire 250 epochs of training with the dropout rate of 0.3 (since it performed the best) to see training and validation curves and check the test accuracy.





The table below compares the results with and without regularization after training on the training data for 250 epochs.

Accuracies	**Without regularization**	**With Regularization**
Train	1	1
Validation	98.4	0.9868
Test	98.2	0.9857

Clearly, both the validation and testing accuracies increase after applying regularization, and so we can conclude that regularization is indeed preventing some overfitting on the training data.

Problem 2

Train a Convolutional Neural Network (CNN) for image classification on the CIFAR-10 dataset.

Visualising Dataset:

Pre-processing dataset

Preprocessing was done in a similar fashion as the Problem 1 on MNIST dataset

- All the datasets were normalised by dividing all the feature pixel values by 255
- The y values were one-hot encoded to bring them into the appropriate format of the neural network output.
- FInally, the shape of the different datasets were:

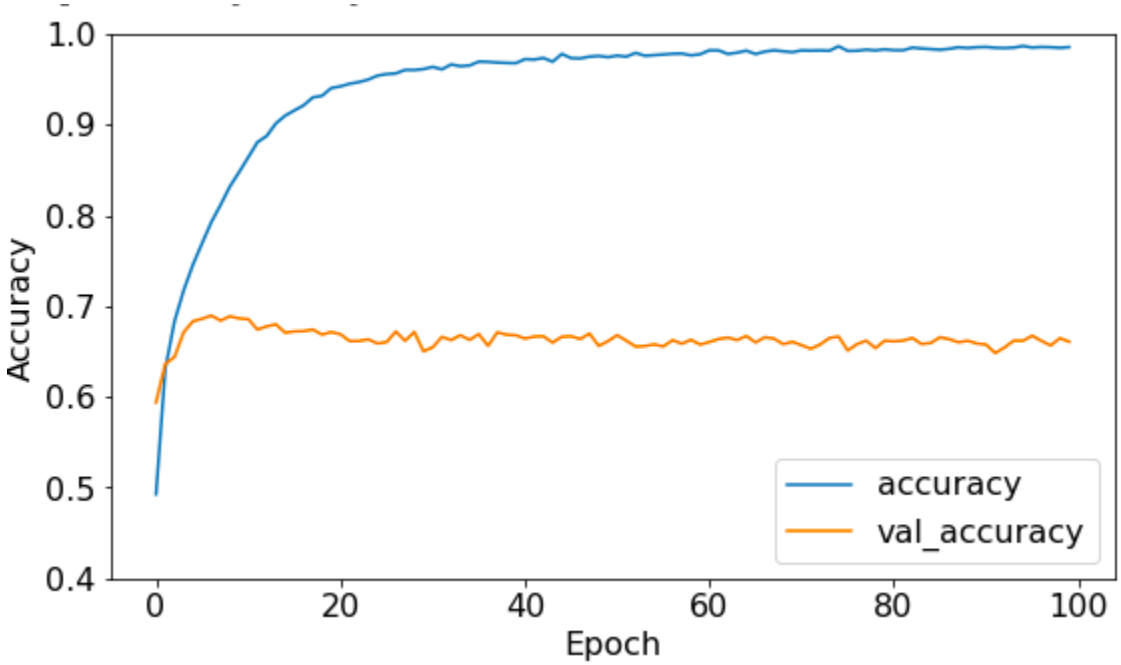
- Train: X=(40000, 32, 32, 3), y=(40000, 10)
- Test: X=(10000, 32, 32, 3), y=(10000, 10)
- Val: X=(10000, 32, 32, 3), y=(10000, 10)

CNN base model used:

For the base model for early testing, we took 2 Conv2D layers, 2 max pool layers and 3 fully connected layers. We applied the basic hyperparameter tuning on this base model and made some modifications wherever required.

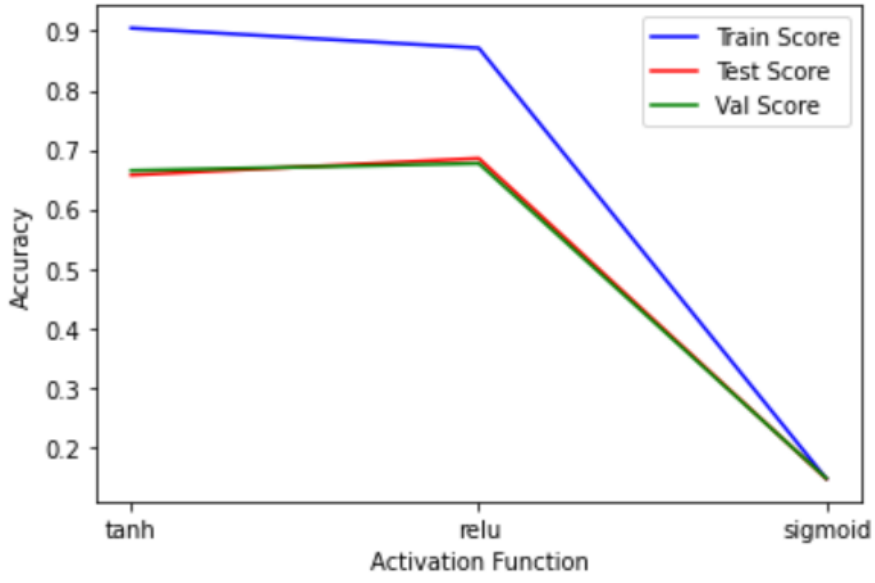
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 4096)              0
_________________________________________________________________
dense (Dense)                (None, 64)                262208
_________________________________________________________________
dense_1 (Dense)              (None, 128)               8320
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 291,210
Trainable params: 291,210
Non-trainable params: 0
```

Varying number of epochs



We can see that the training accuracy crosses 0.9 before 20 epochs and the validation accuracy reaches its saturation well before that. Therefore, on the base model, for hyperparameter tuning, more than 25 epochs isn't required.

Varying activation functions



We can see clearly that ReLU performs extremely well as compared to both tanh and sigmoid activation functions.

Henceforth, we will use ReLU activation function in all the hyperparameter tuning.

Justification

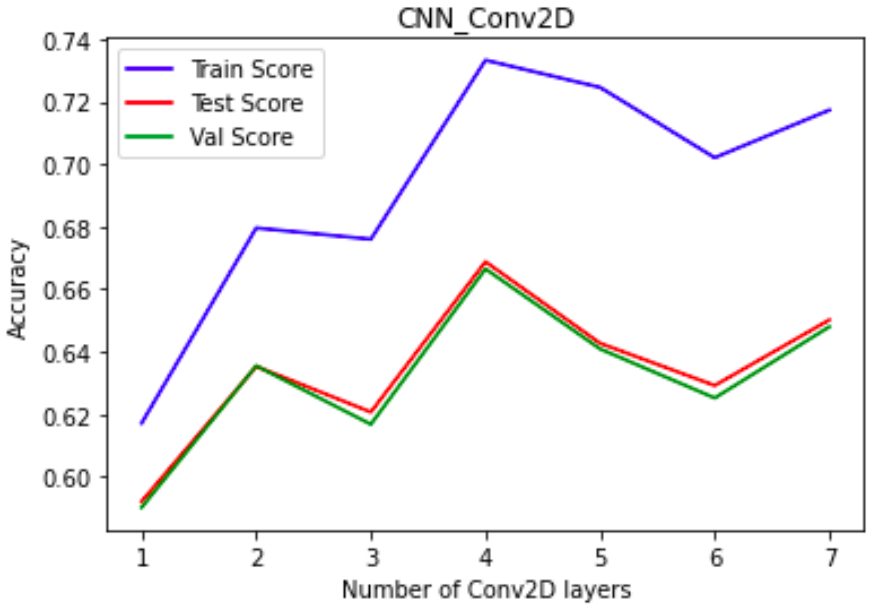
- Derivative of sigmoid (which is used in back propagation for weight update) vanishes for very large values of the activations. Therefore, if the input to the sigmoid is very large, then the weight update is very low. This slows down the learning and often times the algorithm gets stuck in local minimas and so the accuracy isnt high
- This problem of vanishing gradients is absent in ReLU since the derivative is always 1 for positive value of the input.
- Also, ReLU is computationally inexpensive as compared to sigmoid since it is just taking max() function. It's gradient is also simply 1. This is very important for CNNs where there is a lot of computation of the activation function and its gradient. This saves time and computation cost.

Varying number of Conv2D layers

We varied the Conv2D layers in 2 configurations.

CASE 1
Filters in first layer = 32
Filters in layers 2 and 3 = 64
Filters in layer 4 and subsequent layers=128

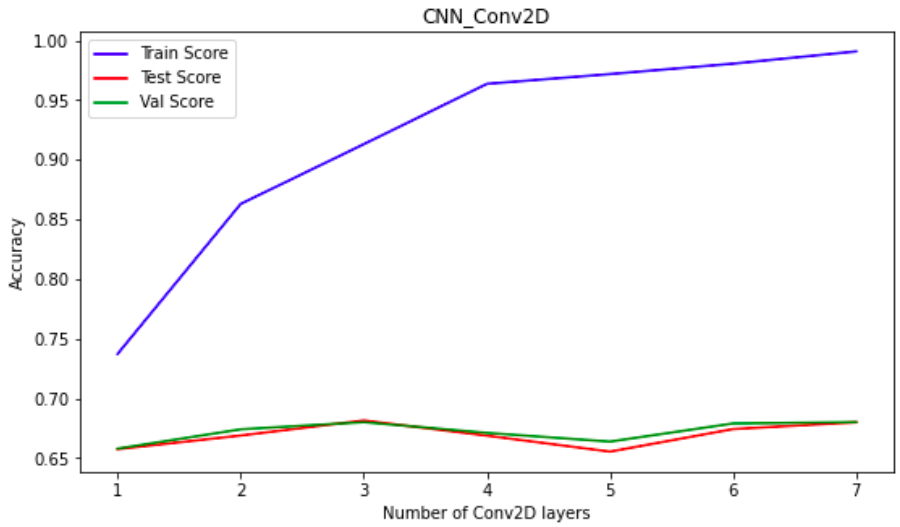
The results on varying the number of layers and training 25 epochs were as follows:



CASE 2
Filters in first layer = 32

Filters in layers 2 and 3 = 64
Filters in layers 3 and 4 =128
Filters in layer 5 and subsequent layers = 256

The results on varying the number of layers and training 25 epochs were as follows:



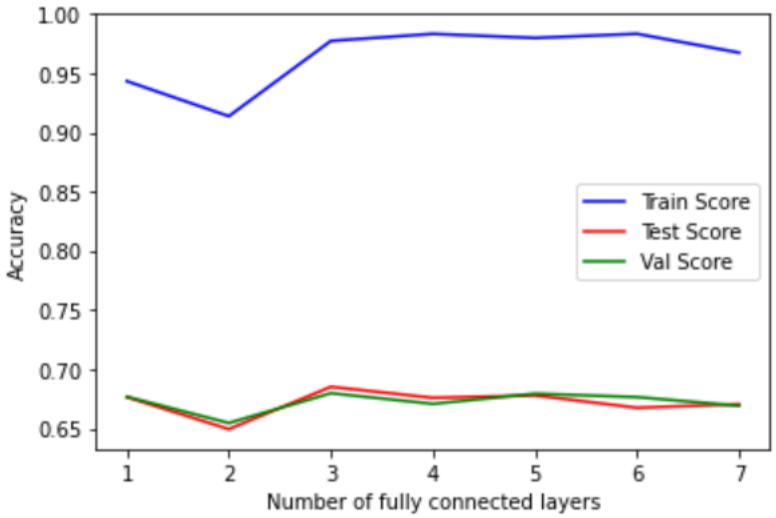
We can see that the configuration in case 2 is clearly better than in case 1, delivering significantly higher training performance. Also, we can see from the plot that overall, increasing the number of Conv2D layers is resulting in better performance. In the final model, we can choose 4-6 layers to achieve a fine balance between training time and accuracy values.

Varying number of fully connected(fc) layers

We varied the number of fully connected layers after the Conv2D layers and the last output layer as follows:

Layer 1: 256 neurons (compulsory)
Layer 2: 256 Neurons
Layer 3, 4,5 = 128 Neurons
Pre-final layer before output layer = 64 neurons

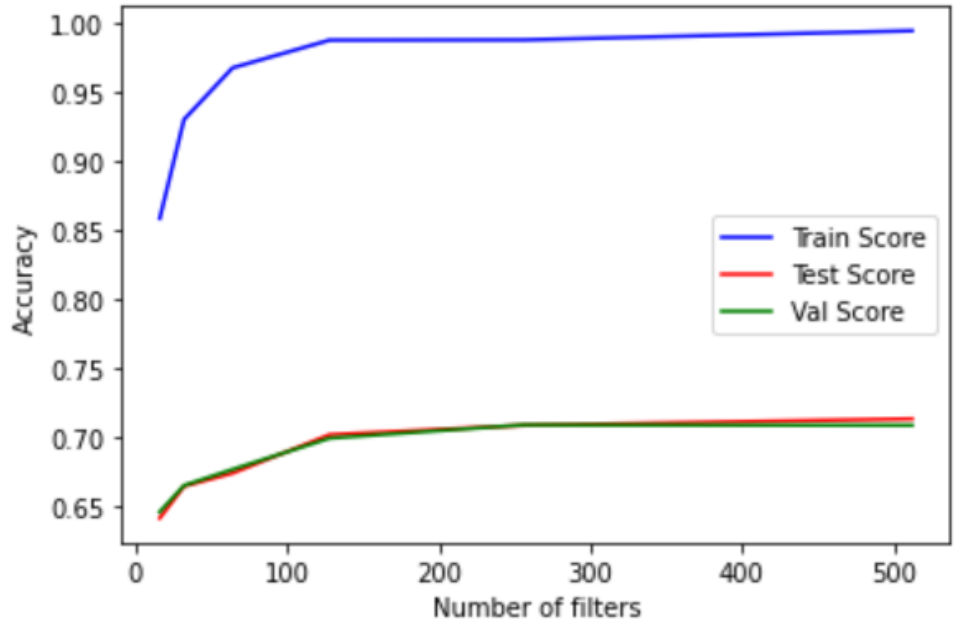
The results on training 25 epochs are as follows:



We achieve a good balance of accuracy and training time for 3 fc layers and so we will use this in the final model. Moreover, we can see that as we are increasing the number of fc layers above 3, the accuracies aren't improving. This might be because we are effectively creating a very deep neural network ahead of the convolutional network, making the model more complex which isn't necessarily helping due to limited data and simple dataset.

Varying number of filters in different layers:

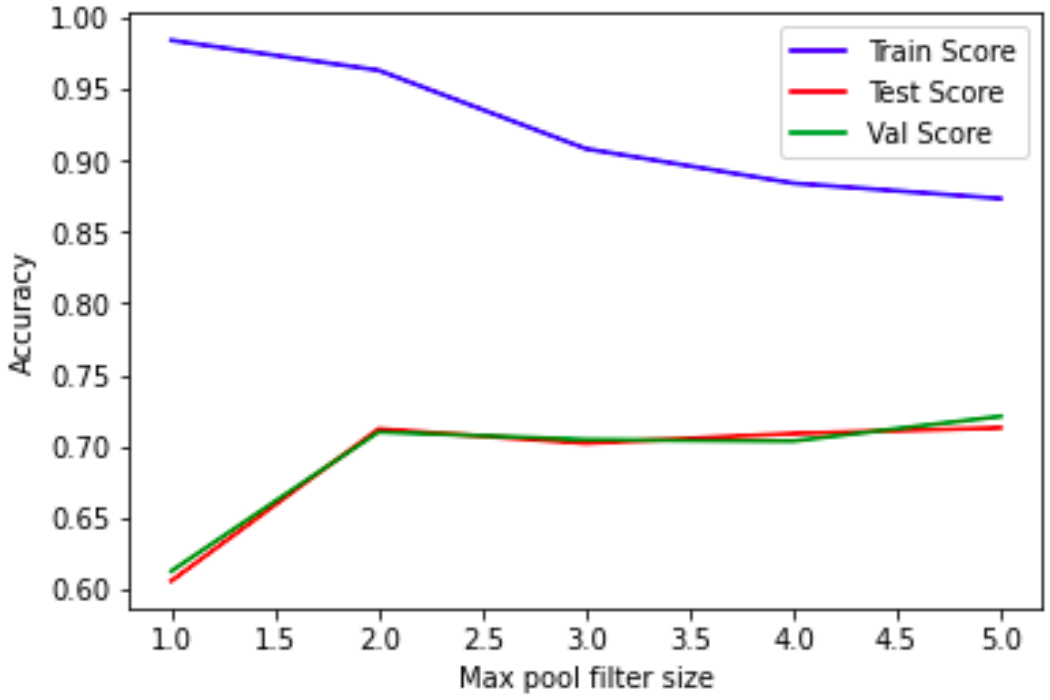
The number of filters used in the Conv2D layers in the base model were varied as [16, 32, 64, 128, 256, 512]. The results on training 25 epochs are as follows:



We can see that 128 and 256 filters are generally good for our purpose. Choosing a very low number of filters isn't desirable since the number of channels in the activations will become very low, which isn't desirable especially in the middle layers of the convolutional network.

Varying Max Pooling filter size

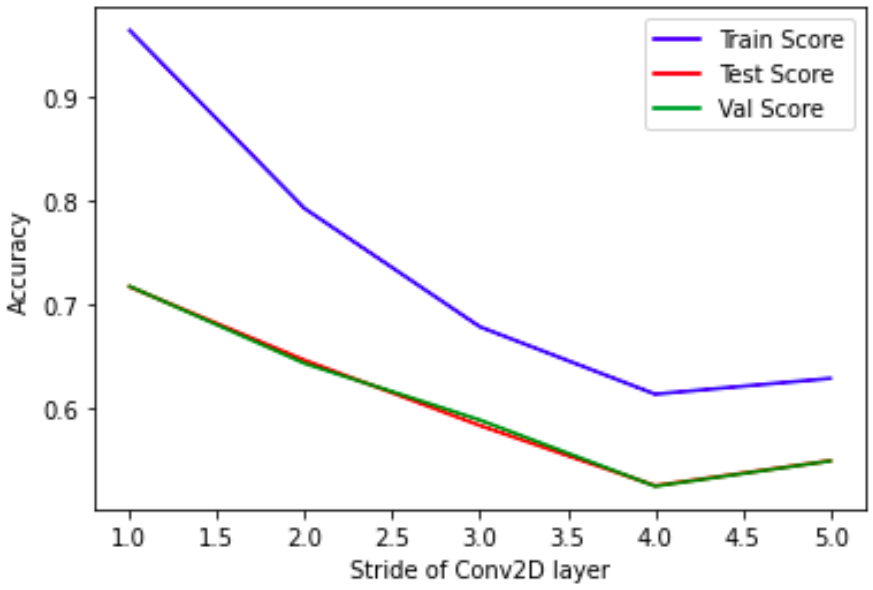
The filter size of max pooling layer was varied between (1,1), (2,2), (3,3), (4,4) and (5,5). The models were trained on 25 epochs and the results are as follows:



The best size of the max pool filter is (2,2) and we get a very good balance of training and validation accuracies.

Varying strides of the conv2D layers

Based on the above results, the max pool layer size is set as (2,2) and the max pool stride is also set as 2, since it is the most common in literature and more stride on max pool will heavily downsample the image which is undesirable for this small network. The strides of the Conv2D layers are varied between 1 to 5 and the models are trained on 25 epochs. The results are as follows:



We can see that as we are increasing the strides, overall the performance of the model is deteriorating. And so ideally we should choose a very small stride value of 1. However, that makes the training computationally very expensive and so we can choose stride of 2 and it is a good balance between efficient downsampling and accuracy.

Summary of varying parameters

Based on our tuning, we summarize the observations as follows:

Parameter	Effect
Epochs	Increasing epochs increases training performance however validation and test performance saturates or can even go down due to overfitting
Activation Functions	ReLU performs best consistently followed by tanh and then finally sigmoid.
Conv2D layers	Accuracy increases with increasing Conv2D layers but so does training time and so a tradeoff is required
Fully connected layers	The validation accuracy saturates after 3 layers but increases upto 3 layers for this specific model
Number of filters	Accuracy saturates after about 256 filters. Different combinations of filters can be used in different layers
Max Pooling	Peak in validation accuracy at pool size of (2,2). This is coherent with most popular implementations of CIFAR10.
Stride	Overall, increasing strides results in poor performance due to severe downsampling which possibly leads to loss of important information

Final Model after tuning

Based on our parameter tuning, we design our final model as an attempt to increase accuracy. After many trials on different combinations of parameters, we settle on this model. Some key model specifications:

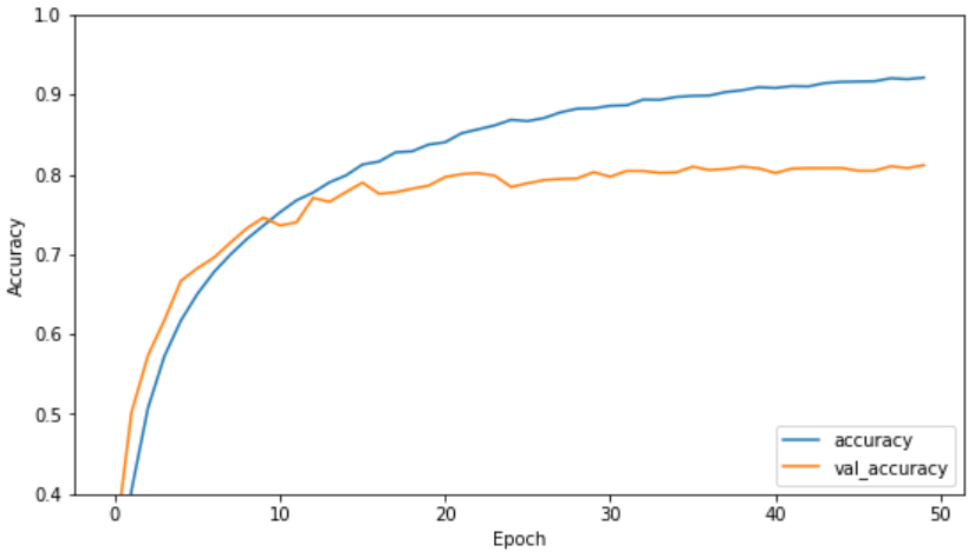
- We will take 5 Conv2D layers as found through our tuning of the number of layers
- The activation functions are taken as ReLU in all the layers
- We have used 3 fully connected layers
- Max pooling filter size has been taken as (2,2) as obtained during tuning

- Number of filters has been varied between 128,256 and 512 in different layers
- Borrowing concepts from the previous problem on MNIST data and through reading literature on CNNs, we decided to use dropout as well after the Conv2D and fc layers. Dropout rate has been taken as 0.3

Model Summary

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 32, 32, 128)	3584
max_pooling2d_2 (MaxPooling2	(None, 16, 16, 128)	0
dropout (Dropout)	(None, 16, 16, 128)	0
conv2d_3 (Conv2D)	(None, 16, 16, 256)	295168
max_pooling2d_3 (MaxPooling2	(None, 8, 8, 256)	0
dropout_1 (Dropout)	(None, 8, 8, 256)	0
conv2d_4 (Conv2D)	(None, 8, 8, 512)	1180160
conv2d_5 (Conv2D)	(None, 8, 8, 512)	2359808
conv2d_6 (Conv2D)	(None, 8, 8, 256)	1179904
max_pooling2d_4 (MaxPooling2	(None, 4, 4, 256)	0
dropout_2 (Dropout)	(None, 4, 4, 256)	0
flatten_1 (Flatten)	(None, 4096)	0
dense_3 (Dense)	(None, 512)	2097664
dropout_3 (Dropout)	(None, 512)	0
dense_4 (Dense)	(None, 256)	131328
dropout_4 (Dropout)	(None, 256)	0
dense_5 (Dense)	(None, 128)	32896
dropout_5 (Dropout)	(None, 128)	0
dense_6 (Dense)	(None, 10)	1290

This model was trained on 50 epochs. The training plot with epochs is as follows:



The accuracy of the final model on the test dataset is **0.8093**

```
313/313 [==============================] - 3s 8ms/step - loss: 0.8822 - accuracy: 0.8093
Test Accuracy: 0.8093000054359436
```

Classification Report

The classification report shows a representation of the main classification metrics on a per-class basis. This gives a deeper intuition of the classifier behavior over global accuracy which can mask functional weaknesses in one class of a multiclass problem. We can see that label 8 and 1, ship and automobile is classified most accurately with a precision of 0.92 and 0.91

	precision	recall	f1-score	support
0	0.84	0.84	0.84	1000
1	0.91	0.92	0.92	1000
2	0.79	0.69	0.74	1000
3	0.60	0.65	0.62	1000
4	0.78	0.80	0.79	1000
5	0.72	0.73	0.72	1000
6	0.84	0.88	0.86	1000
7	0.87	0.84	0.86	1000
8	0.92	0.85	0.89	1000
9	0.86	0.90	0.88	1000
accuracy			0.81	10000
macro avg	0.81	0.81	0.81	10000
weighted avg	0.81	0.81	0.81	10000

Visualizing Misclassifications:

- We can see from the pictures above that our model is performing really well, all the misclassified images can very easily be misclassified by a human as well. They are very similar to the misclassified prediction.
- Some of the horse images have been misclassified because possibly our model learned to predict images of complete horses but fail on close up photos of horses' heads.
- Some of the dogs have been classified as deers and vice versa. This is because of a lot of similarity in the features of dogs and deers.
- Overall, we can see that really closeup photos of animals have been misclassified. If we visualise the dataset (in the first section of this problem), most of the images in the training set have the entire animal or object. Therefore the weights are trained such that these images are correctly classified.