Report

Abbreviations used

Introduction

Input

We use each pixel(normalized) of 2d Gray scale image as an input to our Neural Network. Since, the images from MNIST database are of 28x28 pixels, we have 784 inputs.

Output

The outputs labels from MNIST database are converted to one-hot encoding. From out network, for each class we receive the probability values which we further process it to get various metrics such as accuracy, ROC, Precision-Recall, Confusion Matrix etc. To avoid, unnecessary blotting of the report, we have the plots whenever there's something interesting.

Training Procedure

Out of 70,000 images from MNSIT database, we use 60,000 images as training samples.

Testing Procedure

Out of 70,000 images from MNSIT database, we use 10,000 images as testing samples.

Evaluation Procedure

Since we have multi-class classification problem, we use **categorical_accuracy**. It checks to see if the index of the maximal true value is equal to the index of the maximal predicted value.

NOTE We can use this as loss function as well. We experiment with various loss functions, but we use this as an overall metric to compare models and configurations.

Baseline

We start our study with a very simple network having just one hidden layer

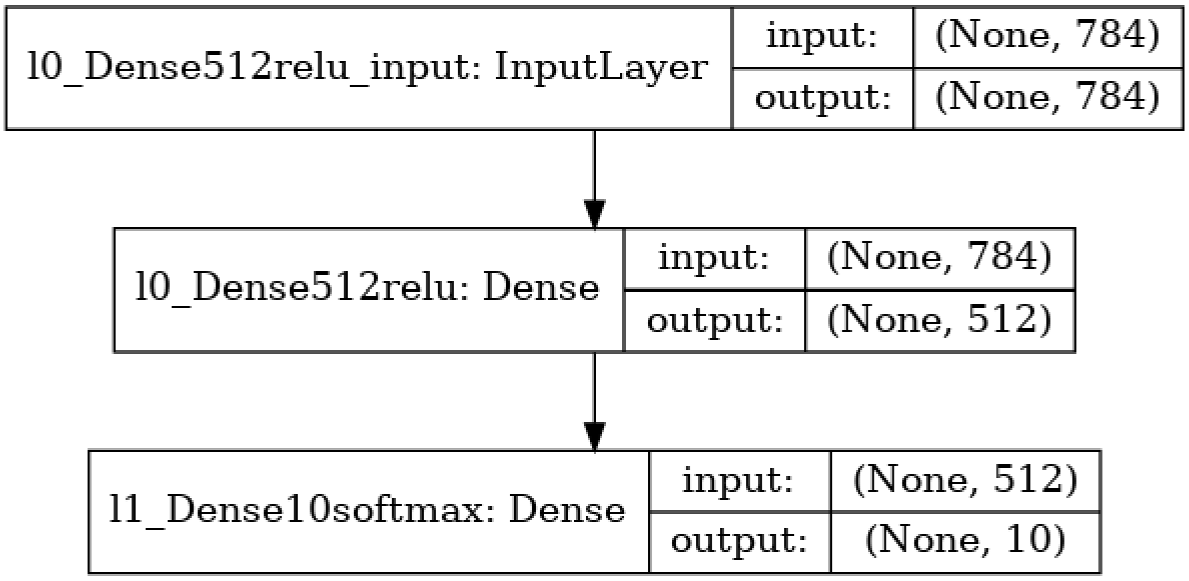

Figure 1. Model Summary

Number of neurons in hidden layer

As starting point, the common literature on online suggests:

```
Number of Outputs <= #neurons <= Number of inputs
#neurons ~ Sum(Number of inputs, Number of Outputs) x 2/3
```

In our case, Sum(Number of inputs, Number of Outputs) x 2/3 = 529.33 ~ 530 Moreover, for reasons of speed, it is advised to use powers of 2. Hence, an ideal value would be: 512. However, to be sure we test with both 256 and 512 neurons

While we evaluate number of neurons, we keep the following parameters constant:

```
{
    "loss_func": "categorical_crossentropy",
    "optimizer": "sgd",
    "activation": "relu",
    "lr": 0.01,
    "nb_epochs": 4,
    "batch_size": 8192
}
```

Results

- 256 Neurons

```
{
    "loss": 2.00770719871521,
    "categorical_accuracy": 0.43059998750686646
}
```

- 512 Neurons

```
{
    "loss": 1.9778928642272948,
    "categorical_accuracy": 0.4708999991416931
}
```

We see a big increase in accuracy by choosing 512 neurons instead of 256 and a reduction in loss. Having more number of neurons means more features can be learnt. To test that hypothesis, when we fed the test data in same order and picked the 1st 9 errors under both the configuration:

- 256 Neurons:

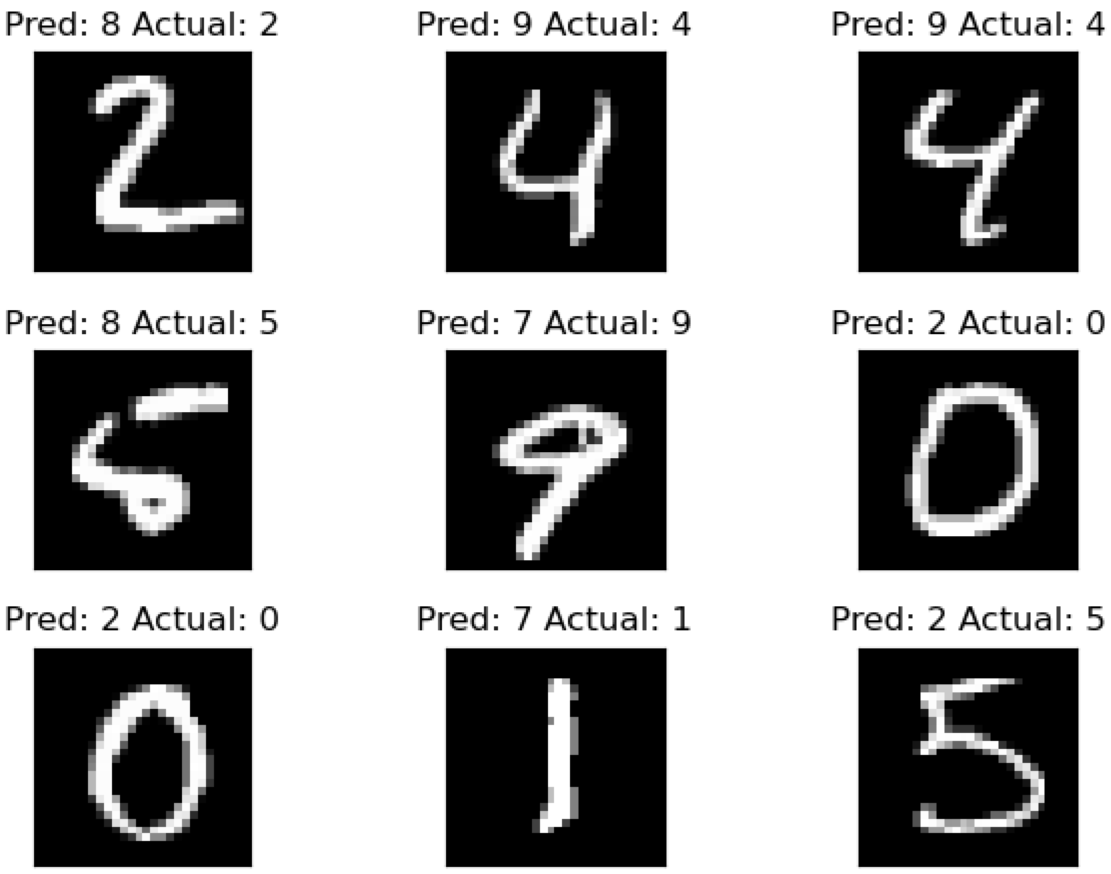

Figure 2. Incorrectly classified samples

- 512 Neurons:

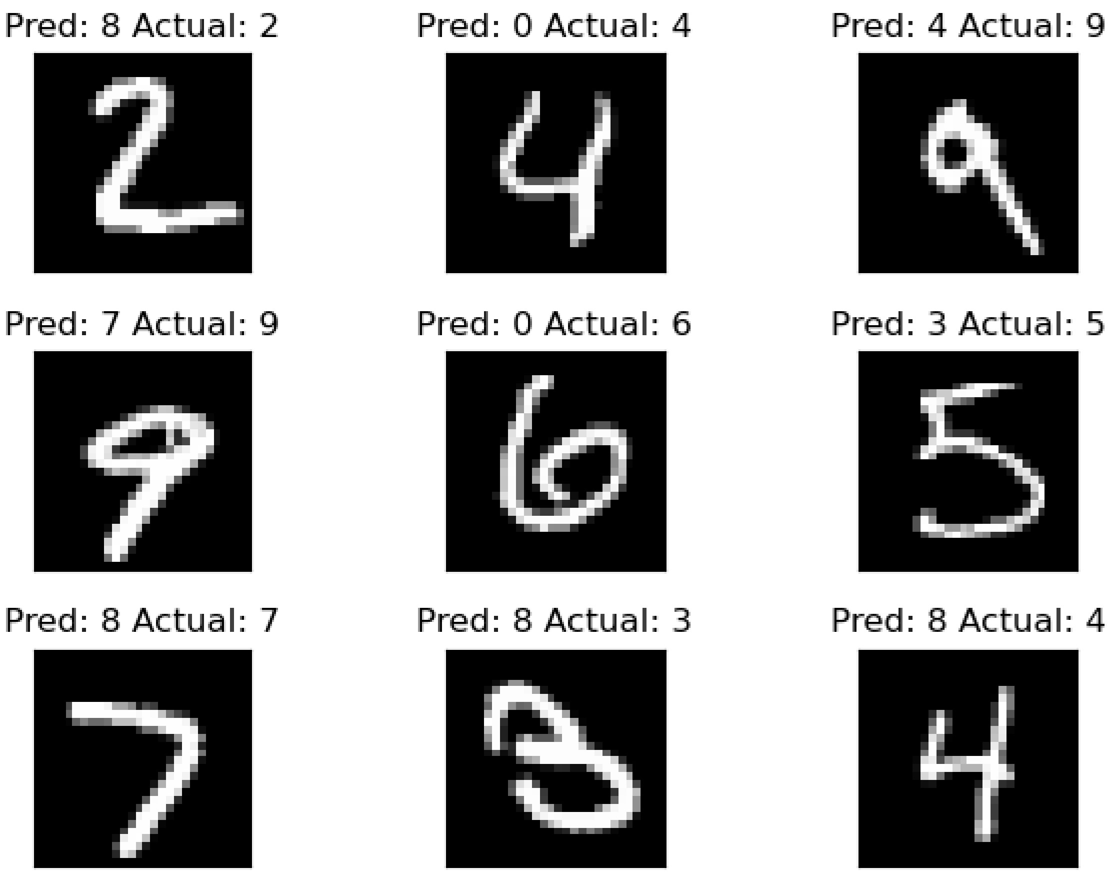

Figure 3. Incorrectly classified samples

As we see 256 neuron network does more errors of same type compared to that of 512.

With 128 Neurons:

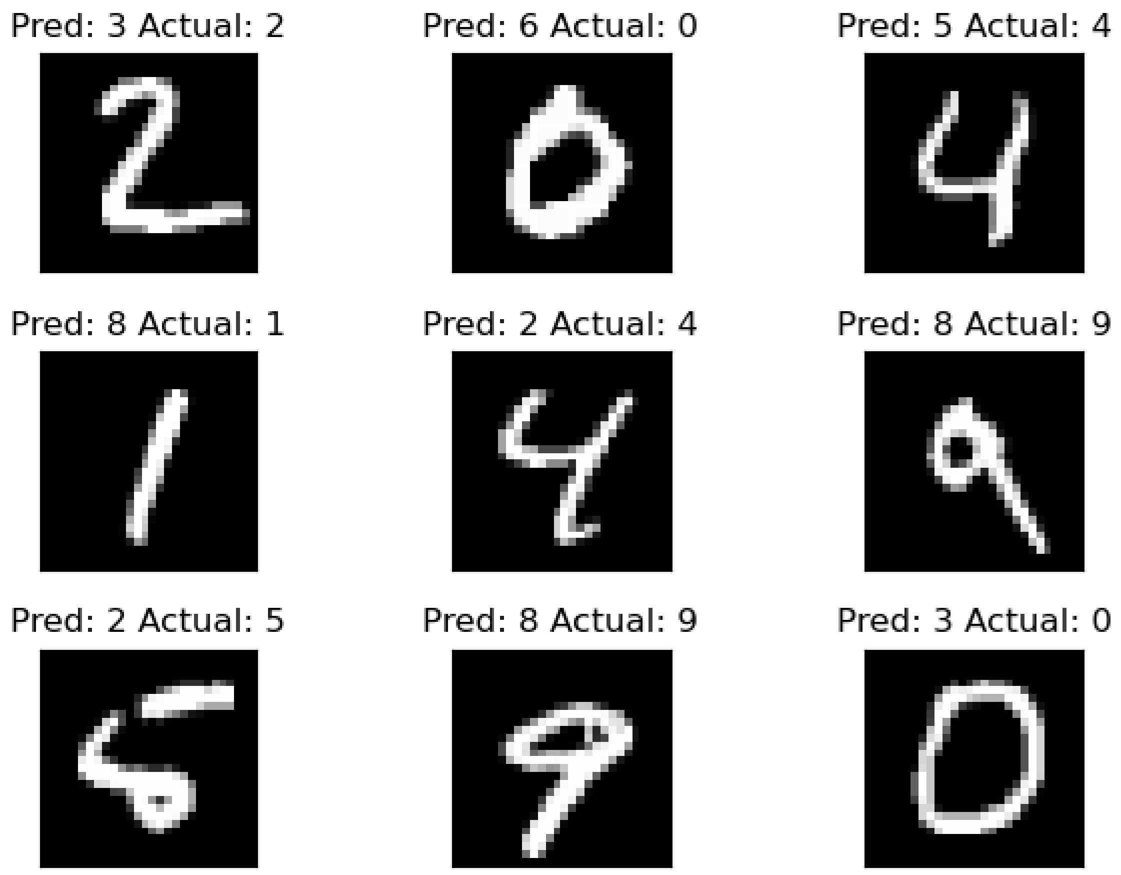

Figure 4. Incorrectly classified samples

We clearly observe that with 128, we find more errors of same type. Therefore, by *having more number of neurons more features can be learnt*. Hence, for **upcoming experiments we use 512 neurons**.

Number of hidden layers

One of the thumb rules used in industry to judge number of hidden layers:

```
0: linearly separable
1: continuous functions
2: arbitrary decision boundary
> 2: complex representations
```

Our data is neither linearly separable nor continuous, hence 1 layer as in last section may not be a good choice. Hence, in this section we try 2, 3 and 4 layers.

For all the tests we keep the following configuration:

```
{
    "loss_func": "categorical_crossentropy",
    "optimizer": "sgd",
    "activation": "relu",
    "lr": 0.01,
    "nb_epochs": 4,
    "batch_size": 8192
}
```

Results

- 2 hidden Layers

```
{
    "loss": 2.0973992691040038,
    "categorical_accuracy": 0.3953000009059906
}
```

- 3 hidden Layers

```
{
    "loss": 2.1838866764068605,
    "categorical_accuracy": 0.34279999136924744
}
```

- 4 hidden Layers

```
{
    "loss": 2.2169689121246337,
    "categorical_accuracy": 0.3668000102043152
}
```

We can see that having anything beyond 2 will reduce the accuracy. In fact, the loss was highest with 4 layers. When we have more layers, the model learns 'fast'(premature) and hence over-fits the data.

If we examine the metrics plot:

- 2 hidden Layers

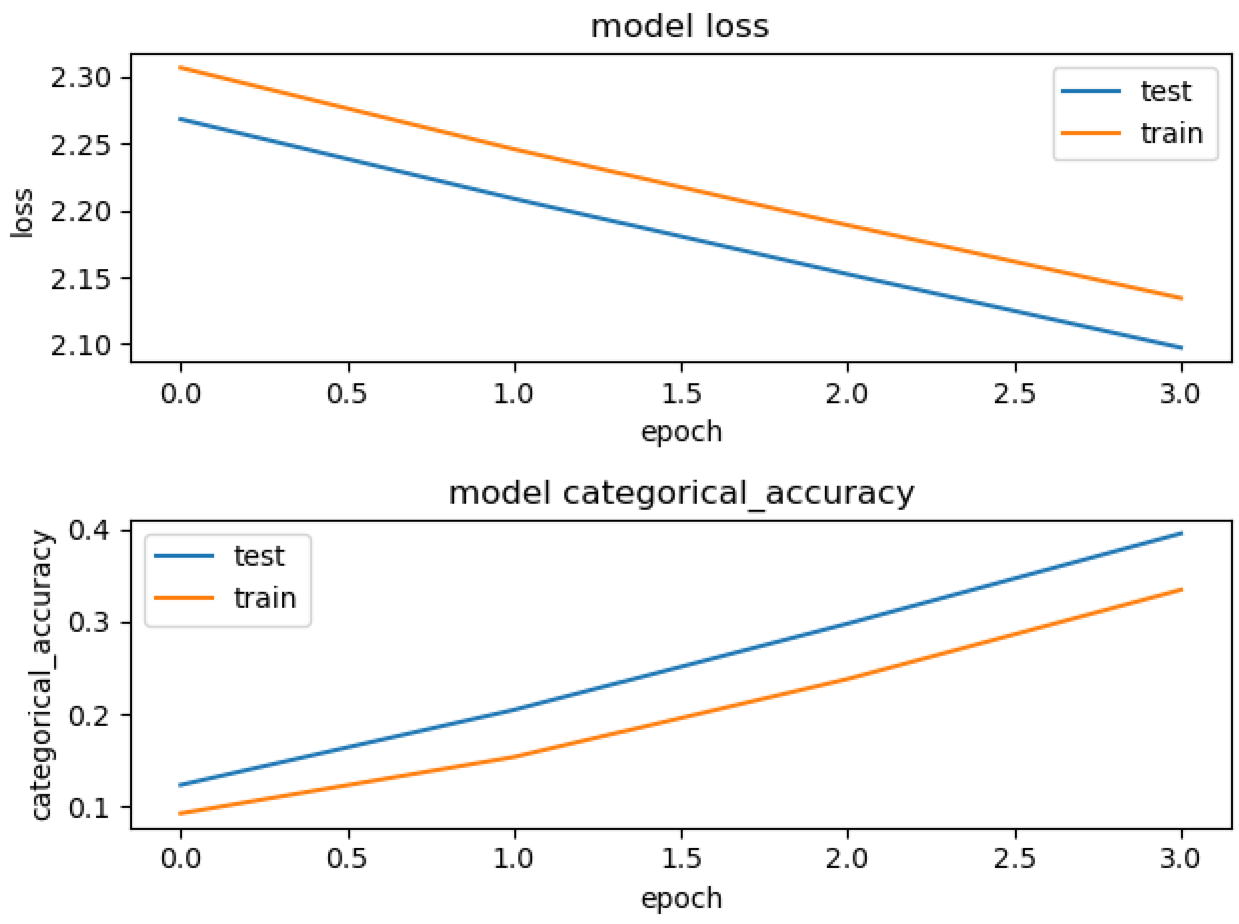

Figure 5. Metrics

- 3 hidden Layers

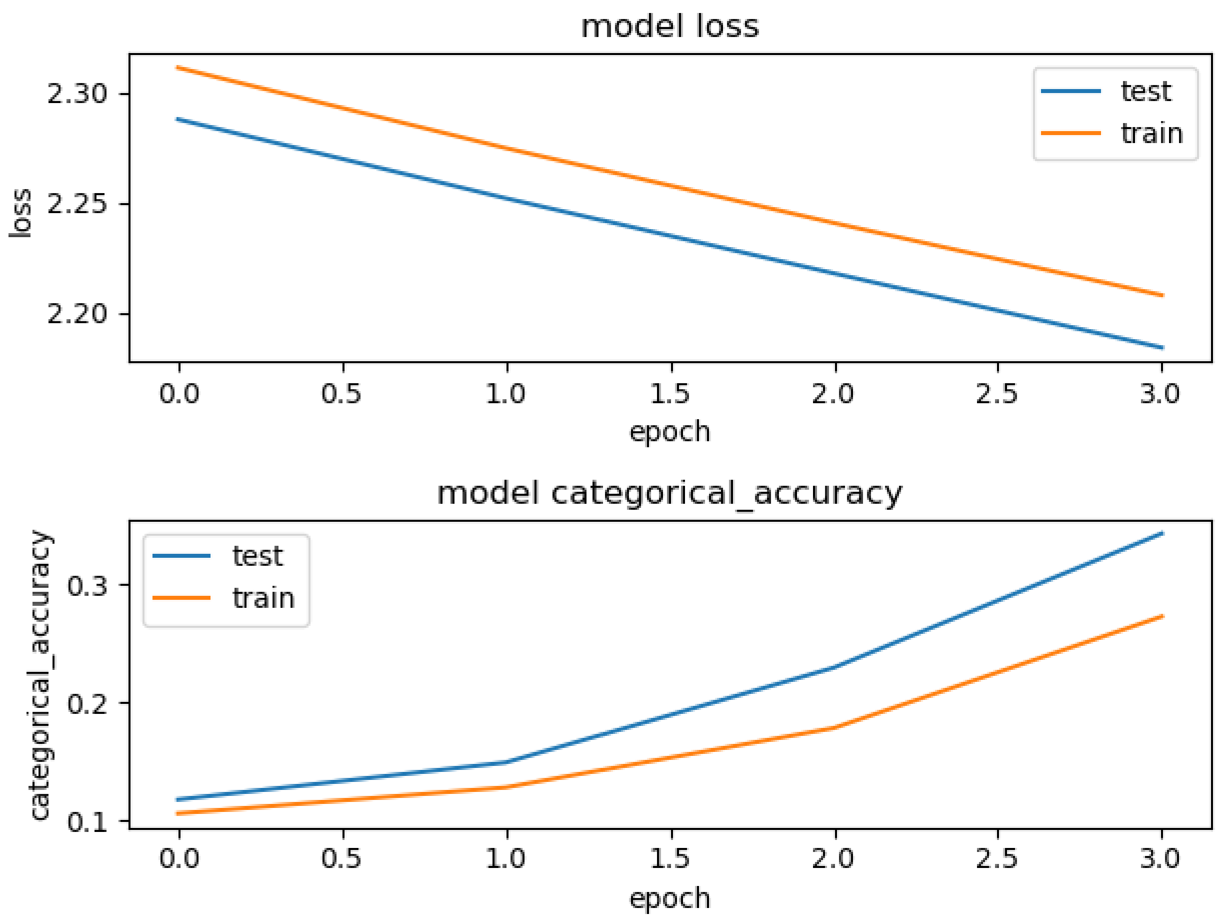

Figure 6. Metrics

- 4 hidden Layers

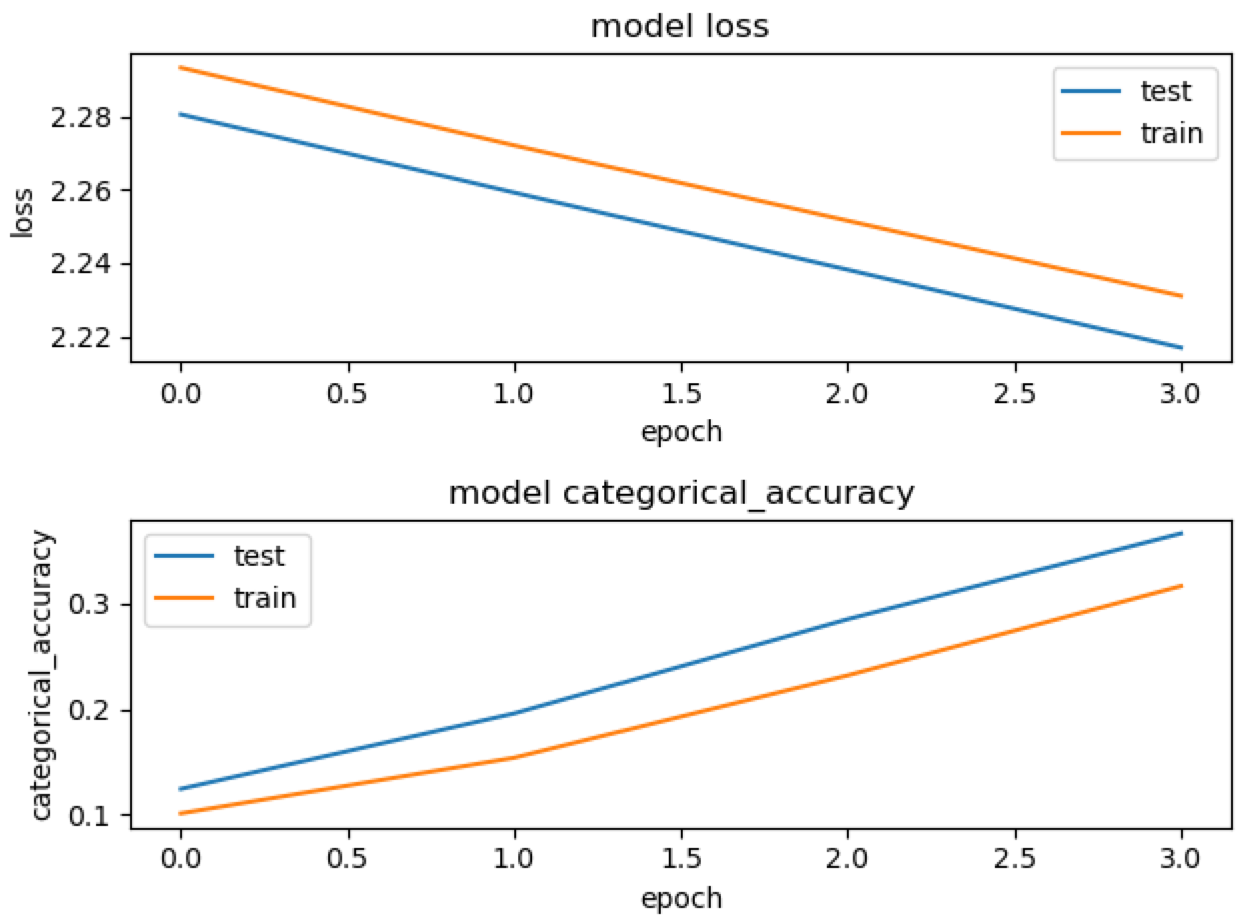

Figure 7. Metrics

If we observe the **Categorical Accuracy** plots, the more the number of layers, sooner and sharper the change in slope occurs. This could indicate that the model is learning too fast and over-fitting.

By increasing the number of epochs over-fitting is bound to happen for models that are under going the problem at the current rate.

We try to check the impact by reducing the number of epochs to 2.

- 2 hidden Layers

```
{
    "loss": 2.1929385009765623,
    "categorical_accuracy": 0.257999986410141
}
```

- 3 hidden Layers

```
{
    "loss": 2.264328922653198,
    "categorical_accuracy": 0.13770000636577606
}
```

- 4 hidden Layers

```
{
    "loss": 2.2645962226867677,
    "categorical_accuracy": 0.15539999306201935
}
```

Even if we reduce the number of epochs, 2 layers perform better. Also, 2 layers is computationally less bulky as well. So, *having more layers will not necessarily improve accuracy.* Therefore, **we will be using 2 hidden layers for all the upcoming tests**

Number of epochs

In the previous section we saw results for 2 hidden layers:

- 2 epochs

```
{
    "loss": 2.1929385009765623,
    "categorical_accuracy": 0.257999986410141
}
```

- 4 epochs

```
{
    "loss": 2.0973992691040038,
    "categorical_accuracy": 0.3953000009059906
}
```

The model and config that we carry from previous section:

```
{
    "loss_func": "categorical_crossentropy",
    "optimizer": "sgd",
    "activation": "relu",
    "lr": 0.01,
    "nb_epochs": 2,
    "batch_size": 8192
}
```

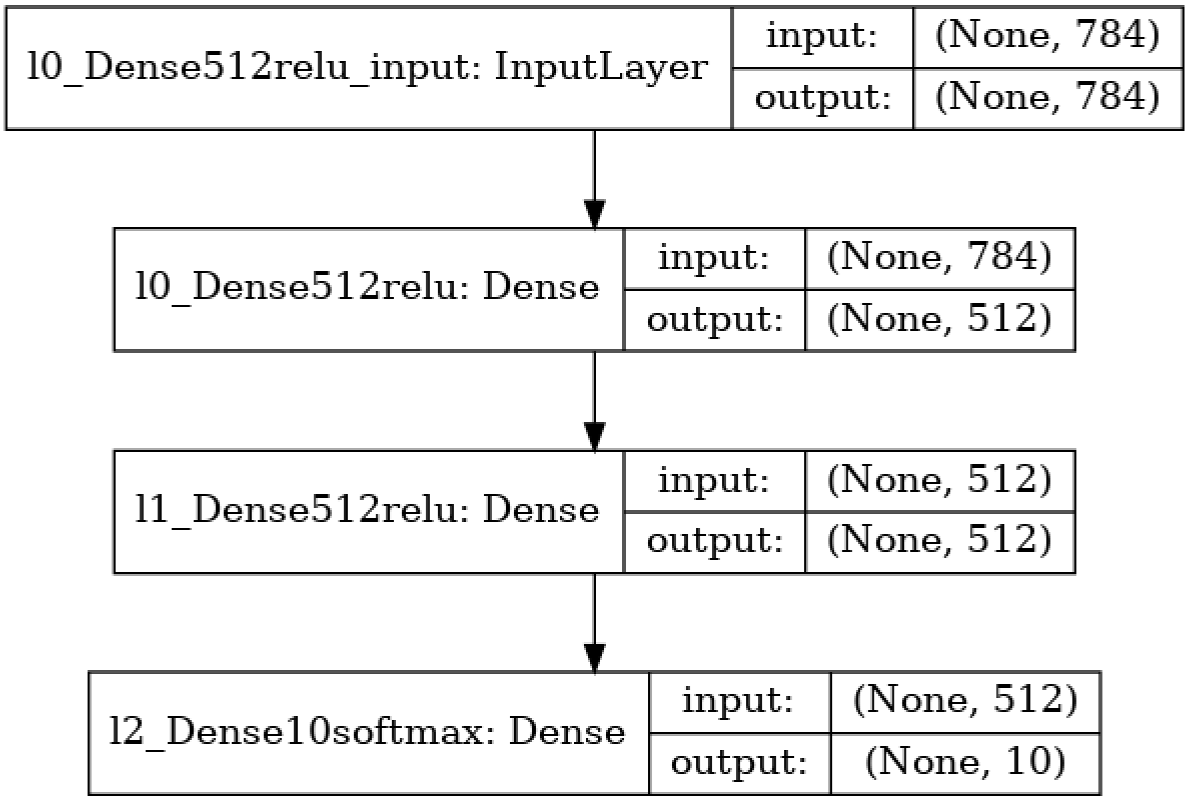

Figure 8. Model Summary

Now under the same configuration, we experiment with 8, 16, 32, 64 .... until we cross atleast 90% accuracy

Results

The plot below is culmination of validation accuracy and loss of networks trained for: 1, 2, 4, 8, 16, 32, 64, 128 epochs

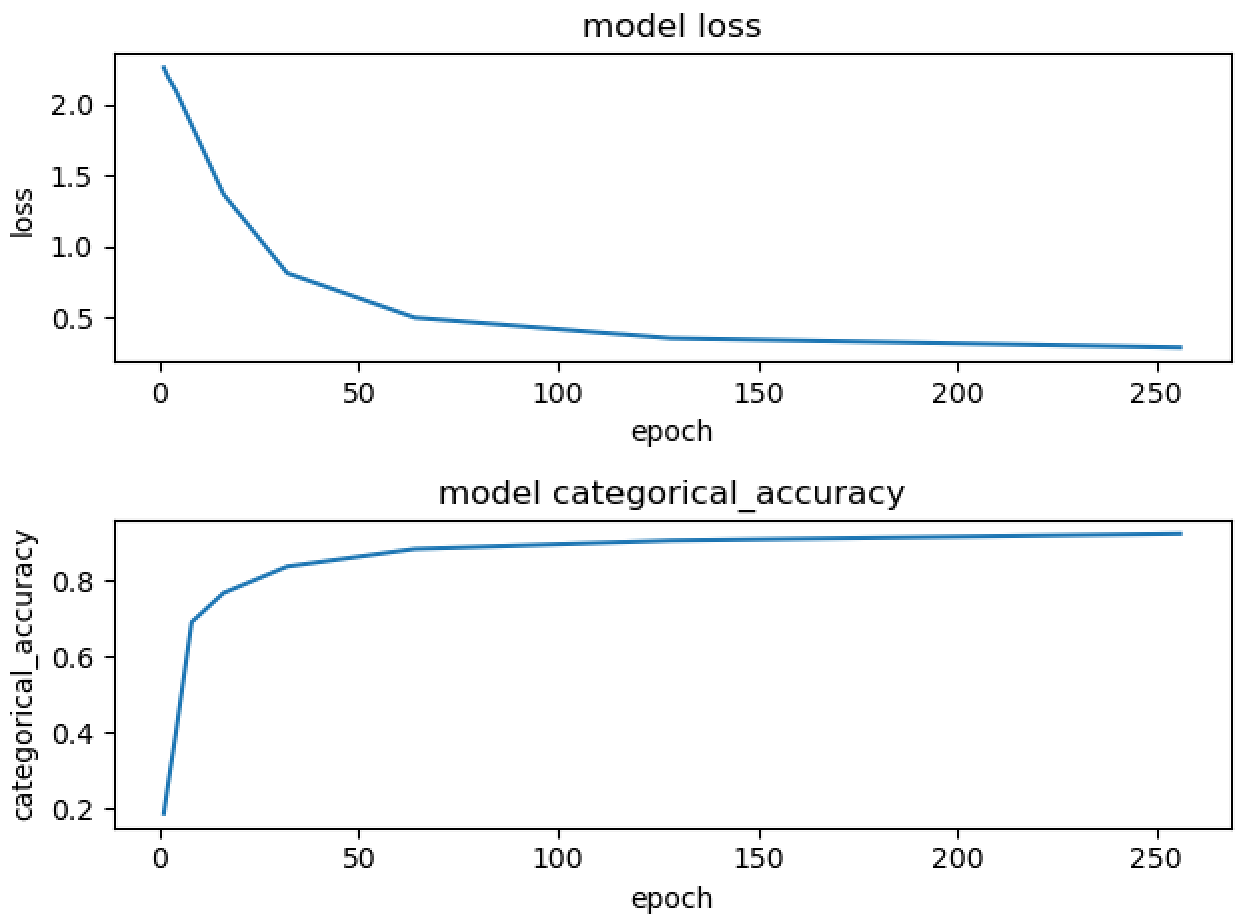

Figure 9. Metrics vs Epochs

As we can see from the graph, though we train the model each time with double the number of epochs as before, we don't necessarily obtain a big increase as we go for higher values. Below is the data of 3 of the models trained under various epochs:

- 64 epochs

```
{
    "loss": 0.4968498522281647,
    "categorical_accuracy": 0.8819000124931335
}
```

- 128 epochs

```
{
    "loss": 0.3520656999707222,
    "categorical_accuracy": 0.9041000008583069
}
```

- 256 epochs

```
{
    "loss": 0.2869773189932108,
    "categorical_accuracy": 0.9222000241279602
}
```

Considering the above trend, to reach accuracy > 95%, we'll need to train model for 1024 or maybe 2056 epochs. So, maybe we can take 2 epochs:64 and 128 & experiment with various batch sizes.