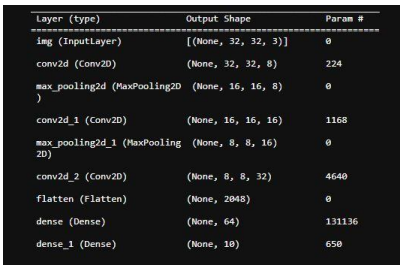
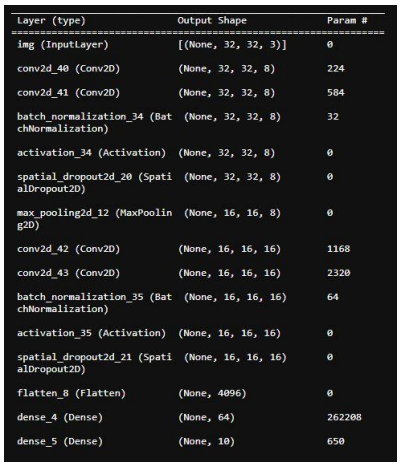
Training and Adapting Deep Networks

Sample is an abbreviated SVHN dataset with 32x32 pixel pictures in three color channels. The training dataset has 10,000 images, whereas the testing dataset contains 1,000 images. On this dataset, a One vs One Linear SVM, a Convolutional Neural Network (CNN) with no data augmentation, a CNN with data augmentation, and ultimately a CNN with VGG architecture algorithms are deployed.

A basic three 2-dimensional convolutional layer with two max-pooling layers in between is utilized for the CNN's backbone in models with and without augmented input. The convolutional networks employ incremental filters with sizes of 8, 16, and 32, respectively, while the pooling layer employs a filter with a size of 2x2. The final convolutional layer is then passed via a flattened layer, followed by a final dense layer with relu activation and an output layer with softmax activation and 10 class output since the dataset has 10 labels for SVHN. A somewhat more sophisticated model with 15 layers is built for the model that fine-tunes a VGG network.



Layer (type)	Output Shape	Param #
img (InputLayer)	[(None, 32, 32, 3)]	0
conv2d (Conv2D)	(None, 32, 32, 8)	224
max_pooling2d (MaxPooling2D)	(None, 16, 16, 8)	0
conv2d_1 (Conv2D)	(None, 16, 16, 16)	1168
max_pooling2d_1 (MaxPooling 2D)	(None, 8, 8, 16)	0
conv2d_2 (Conv2D)	(None, 8, 8, 32)	4640
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 64)	131136
dense_1 (Dense)	(None, 10)	650



Layer (type)	Output Shape	Param #
img (InputLayer)	[(None, 32, 32, 3)]	0
conv2d_40 (Conv2D)	(None, 32, 32, 8)	224
conv2d_41 (Conv2D)	(None, 32, 32, 8)	584
batch_normalization_34 (Bat chNormalization)	(None, 32, 32, 8)	32
activation_34 (Activation)	(None, 32, 32, 8)	0
spatial_dropout2d_20 (Spati alDropout2D)	(None, 32, 32, 8)	0
max_pooling2d_12 (MaxPoolin g2D)	(None, 16, 16, 8)	0
conv2d_42 (Conv2D)	(None, 16, 16, 16)	1168
conv2d_43 (Conv2D)	(None, 16, 16, 16)	2320
batch_normalization_35 (Bat chNormalization)	(None, 16, 16, 16)	64
activation_35 (Activation)	(None, 16, 16, 16)	0
spatial_dropout2d_21 (Spati alDropout2D)	(None, 16, 16, 16)	0
flatten_8 (Flatten)	(None, 4096)	0
dense_4 (Dense)	(None, 64)	262208
dense_5 (Dense)	(None, 10)	650

Image 1: Simple CNN backbone

Image 2: CNN with VGG architecture

I incorporated a pre-trained 2-stage CIFAR-10 small dataset with a total of 267,250 parameters. Four 2d convolution layers, two batch normalization layers to minimize overfitting, spatial dropouts, and two dense layers comprise the tuned VGG network. Adam, a stochastic gradient descent method, is used as the choice of optimiser with the labels set to log tensors.

Both convolutional networks are maintained relatively shallow in depth. All of the images are used as-is, with no image color channel conversion or rescaling. This is done while keeping the test data's magnitude in mind and maintaining all image information. The augmented data was obtained by Keras preprocessing. Random batches of photos were vertically flipped, rotated randomly, and zoomed by minor angles. To keep the constants uniform across all models, batch sizes of 32 and epochs of 100 are employed for all three neural networks.

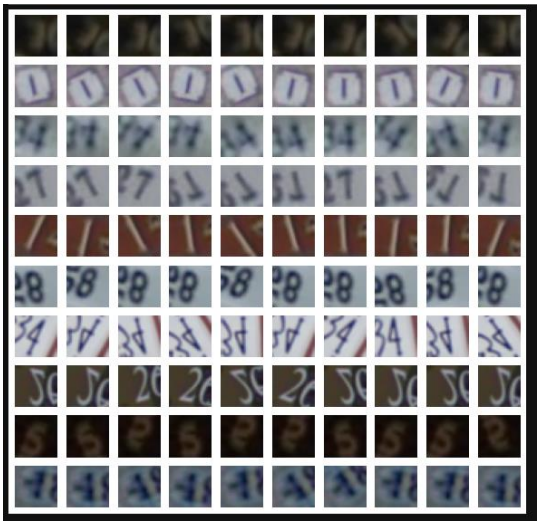


Image 3: Augmented Data

The processing is handled by an Intel Core i7 4702MQ processor with a basic clock speed of 2.20 GHz and an Nvidia GeForce GT740M graphics card. Tensorflow does not support the requisite CUDA toolkit (3. x) since the GPU is outdated. As a result, all computation is done on the CPU. I wish I had a more powerful GPU.

Models	Test Accuracy	Test Loss	Training Time (in sec)	Evaluation Time (in sec)
1v1 Linear SVM	35.62%		32.73	5.94
CNN	70.41%	3.00	6.95	191.89
CNN with Data Augmentation	60.37%	1.59	6.62	218.5
CNN with VGG tuning	70.20%	2.15	12.03	445.29

Table 1: Model statistics

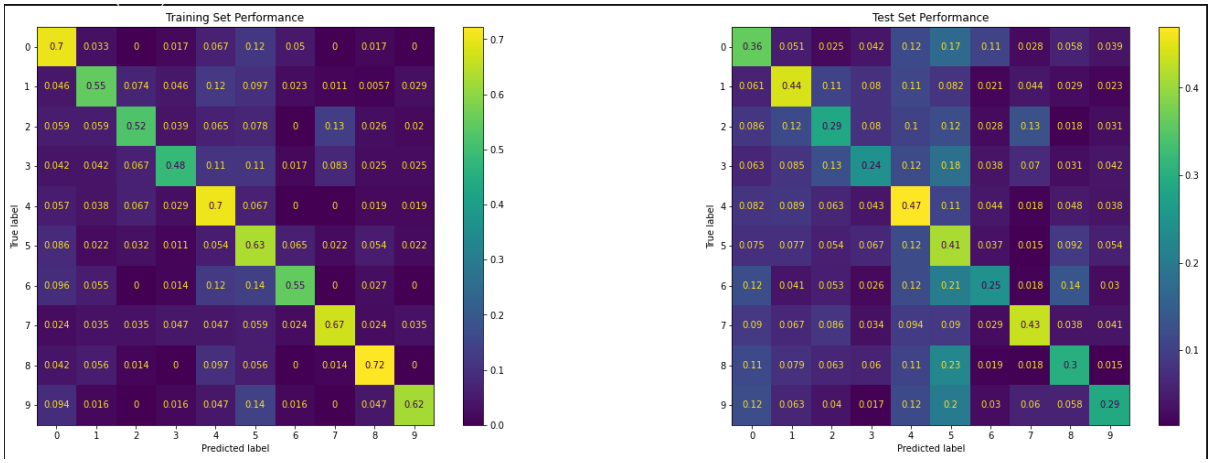


Image 4: Confusion Matrix of 1v1 Linear SVM model

On 1v1 Linear SVM, overall performance was poor. The model exhibits overfitting, and classification accuracy declines in all classes. Perhaps low performance is the result of a class imbalance.

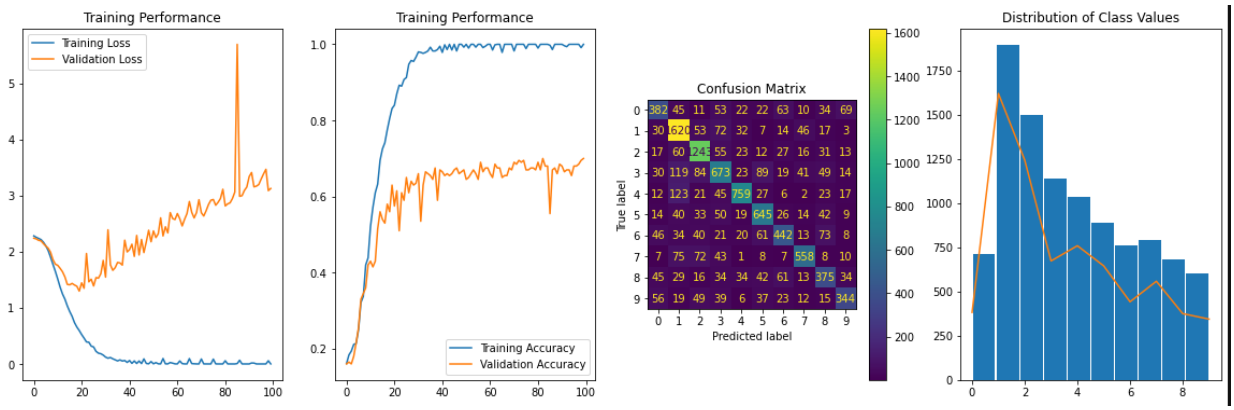


Image 5: CNN model results

We can see overfitting in the CNN model as well, however, the model has converged with good accuracy and a healthy computational time. Also, the model seems to be influenced by noise in the validation dataset.

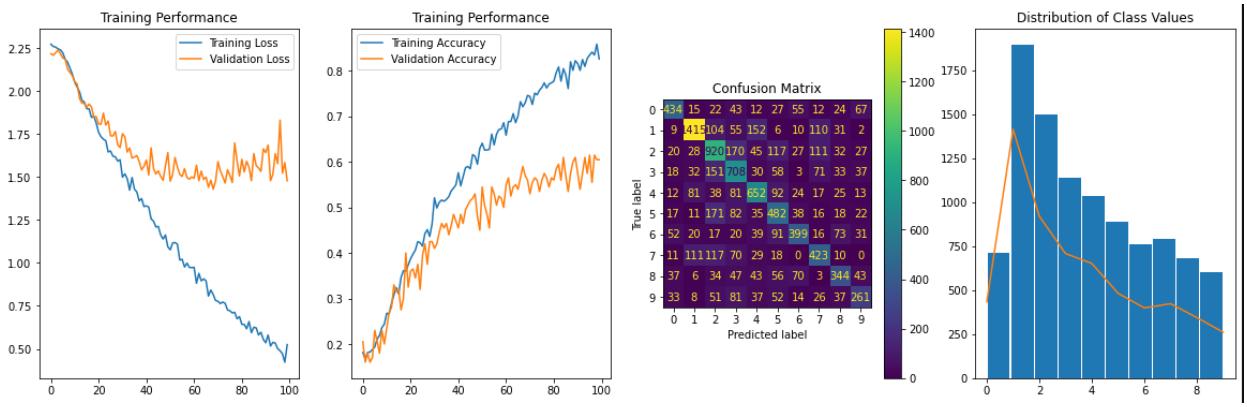


Image 6: CNN model with data augmentation

Training loss on the CNN model with data augmentation is still not converged and the model accuracy has decreased when compared to the CNN m model with any data augmentation. Increasing the epochs and allowing the model to converge might drastically increase the model accuracy with data augmentation.

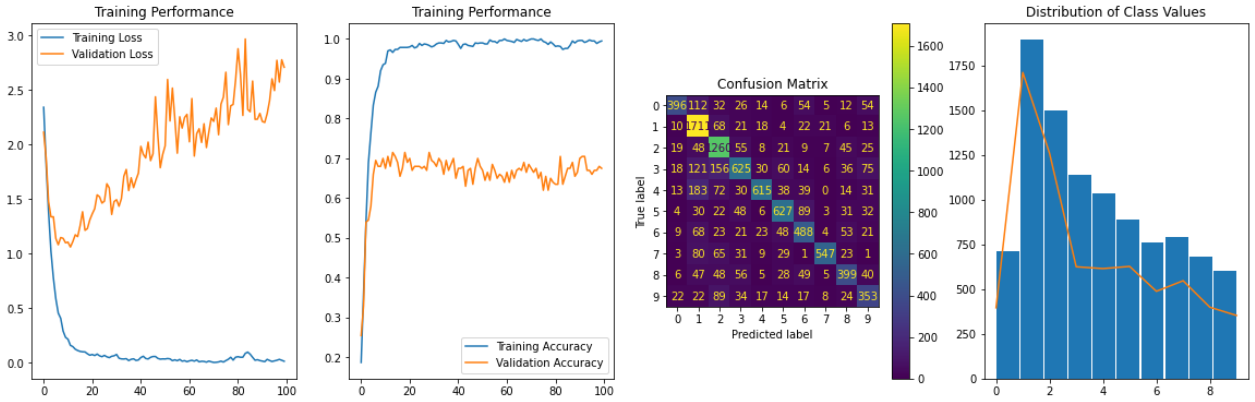


Image 7: CNN network with VGG architecture

Though the model accuracy is similar to a simple CNN model, the validation loss is lesser. The dataset size and the nature of images would contribute a lot towards the performance of these neural networks and having more testing datasets or ‘new’ data will help these models learn more features and perform better.

In the entirety of the performance of these models, the simple CNN model performed better than the rest of the models.