CIFAR-10 Dataset Image Classification using Convolutional Neural Networks

Ashita Chandnani
ECE 697 Fall 2020

Background

Convolutional Neural Networks (CNNs) are fundamental building blocks for image classification tasks to address bigger problems in Computer Vision, Face Recognition System and Self-Driving Cars etc. The performance of such networks largely depends on the depth of the network. This project aims at implementing a deep CNN to perform image classification on the CIFAR-10 (Canadian Institute For Advanced Research) dataset. This dataset is loaded from Keras datasets module. It comprises of 60000 color images with size 32x32. The training and test sets consists of 50,000 and 10,000 images respectively. We choose 5,000 training images for validation data. The model is trained to classify the images into ten classes as follows: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. CIFAR-10 is a very well know dataset and has been extensively studied for image classification problems. Huang et al. achieved an accuracy of 96.54% on this dataset [1].

Project Goal

In order to introduce non-linearity, the most widely used activation function is Rectified Linear Unit (Relu). Ramachandran et al. proposed a new activation function named Swish which tends to perform better than Relu for training deeper networks [2]. It is simply a gated version of the sigmoid activation and is given by $f(x) = x \cdot \text{sigmoid}(x)$. This project aims to implement a deep neural network to train on CIFAR-10 dataset by using Keras/Tensorflow code inspired from Chollet and Neilson[3],[4]. Since the training of deep networks require huge computing power and long training times, the codes are run on BSU R2 cluster and also on Google Colab which is a hosted Jupyter notebook service and provides free access to GPUs. The new ideas/approaches include:

a. Investigate and benchmark different network architectures by varying number of convolution layers, different number of filter sizes, different number of pooling layers etc
b. Different activation functions beyond conventional RELU e.g. Swish
c. Optimizing training parameters like learning rate
d. Further, to improve test accuracy and reduce overfitting on the training data, dropout and regularization are employed.
e. Results are summarized. Optimum network, activation, training parameters are identified based on the above design of experiments.

Dataset

- The CIFAR-10 dataset is loaded from the standard Keras datasets.
- The dataset comprises of 60000 color images (size 32*32*3) in 10 classes, with 6000 images per class.
- There are 50000 **training** images and 10000 **test** images.
- The original training data (50,000 images) is split into **90% training** (45,000 images) and **10% validation** (5,000 images) data
- Resulting training-validation-test data split ratio is 4.5: 0.5: 1.0 over the total of 60,000 images.
- The test batch contains exactly 1000 randomly-selected images from each class.
- As part of data preparation, the data is normalized between 0 and 1 to avoid slow or unstable learning process.

Design Methodology

- Several CNN models are created using Keras framework by varying the **number of convolution and pooling layers** with two different **filter sizes**.
- Two fully connected layers with 1024 neurons each are followed by 10 neurons in the output layer corresponding to the 10 desired classes.
- For all models, **Adam** optimizer and **Categorical cross entropy loss** is employed because this is a multi-class classification problem where each example belongs to a single class.
- All the trainings employ a **batch size** of 32 with the number of **epochs** set to 25.
- The final output layer employs Softmax activation. All other layers employ either **Swish or Relu** activations in order to investigate and make a comparison between the two.
- The problem of overfitting is removed by incorporating **dropout** and L2 **regularization**.
- Further improvement in validation and test accuracy is made by incorporating **batch normalization** to yield a final CNN model with an accuracy of 85.21% on test data.
- P**adding** (which adds a one pixel border to the image with a pixel value of zero) is employed in all the convolution layers. This allows for more space for the kernel to cover the image leading to a more accurate analysis of images. Because of this the size of the

convolved maps is same as the size of the input to the convolution layer. E.g. in this project, the size of the convolved maps is calculated as = input + pad-kernel +1.

E.g 32+2-3=1=32 (for 1st layer).

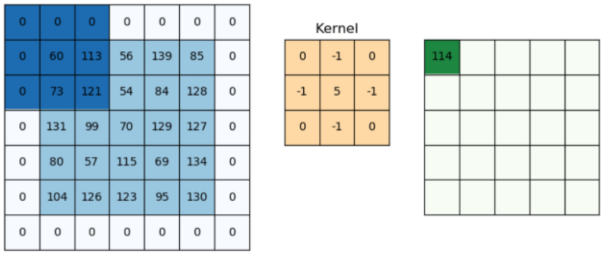


Figure 1: A *3×3* kernel applied to an image with "same" padding [5]

A careful design of experiments and a step by step analysis to obtain the final architecture is given below.

A. Varying kernel size

To begin with, two different filter sizes (3x3 and 5x5) with a stride of 1 are evaluated for architectures with 4 and 5 convolutional layers followed by max pooling layers. 2×2 max pooling layers with stride of 2 are used. Learning rate is set to 0.0005 to be. Dropout is not employed here. The set of experiments are summarized below.

Experiment number 1	**Vary filter sizes for 5 layer CNN and 4 layer CNN with Relu and Swish**							
Number of Conv layers	5	5	5	5	4	4	4	4
Filter size	3x3	3x3	5x5	5x5	3x3	3x3	5x5	5x5
Activation Function	Relu	swish	Relu	swish	Relu	swish	Relu	swish
Conv layer 1-Pool1	128	128	128	128	128	128	128	128
Conv layer 2-Pool2	256	256	256	256	256	256	256	256
Conv layer 3-Pool3	256	256	256	256	256	256	256	256
Conv layer 4-Pool4	512	512	512	512	512	512	512	512
Conv layer 5-Pool5	512	512	512	512	-	-	-	-
Dense Layer 1 size	1024	1024	1024	1024	1024	1024	1024	1024
Dense Layer 2 size	1024	1024	1024	1024	1024	1024	1024	1024
Output Layer size	10	10	10	10	10	10	10	10
Test Accuracy	**74%**	**75%**	**72%**	**74.10%**	**74.36%**	**74.27%**	**71.49**	**73.54%**

- Based on above results a smaller kernel size (3*3) performs better in this classification problem.
- Smaller kernel size better captures complex non-linearities in the input. Variations in data are better sustained because lesser number of pixels are averaged every time.
- Performance of Swish is comparable to Relu activation.
- More the number of convolution layers better is the accuracy however system complexity increases for a bigger network.
- Accuracy is in range 74%-75% for the above simple networks.

Thus, filter size of 3*3 was chosen for final architecture.

B. **Including dropout and regularization**

Architectures with 4 and 5 convolutional layers followed by max pooling layers (2×2) stride of 2 are used. Learning rate is set to 0.0005. Dropout of 0.2 after every pool and dense layer (except output) and L2 regularization (default 0.00001 on both dense layers) are employed here. The set of experiments are summarized below.

Experiment number 2	**Including dropout and regularization**			
Number of Conv layers	5	5	4	4
Filter size	3x3	3x3	3x3	3x3
Activation Function	Relu	swish	Relu	swish
Conv layer 1-Pool1	128	128	128	128
Conv layer 2-Pool2	256	256	256	256
Conv layer 3-Pool3	256	256	256	256
Conv layer 4-Pool4	512	512	512	512
Conv layer 5-Pool5	512	512	-	-
Dense Layer 1 size	1024	1024	1024	1024
Dense Layer 2 size	1024	1024	1024	1024
Output Layer size	10	10	10	10
Test Accuracy	**79.96%**	**79.49%**	**80.40%**	**79.72%**

- With filter size 3*3, the above 4 CNNs clearly show an improvement in accuracy by including dropout and regularization.
- Regularization and dropout help to reduce overfitting on the training data.
- Accuracy goes up to 79-80% with *dropout* compared to 74-75% without dropout.

C. Varying number of pooling layers and convolution layers (With dropout)

Architectures with 4,5,6 convolutional layers followed by max pooling layers (2×2) stride of 2 are used. Learning rate is set to 0.005. Dropout of 0.2 after every pool and dense layer (except output) and L2 regularization (default 0.00001 on both dense layers) are employed here. Here three architectures with number of convolution layers 4, 5 ,6 are evaluated. Also, the effect of removing 1st pooling layer is studied as well as effect of removing alternate pooling layers is investigated. The set of experiments are summarized below.

Experiment number 3	**Effect of varying number of conv and pooling layers**					
Number of Conv layers	6	6	5	5	4	4
Filter size	3x3	3x3	3x3	3x3	3x3	3x3
Activation Function	Relu	swish	Relu	swish	Relu	swish
Conv layer 1 – No Pool1	128	128	128	128	128	128
Conv layer 2-Pool2	128	128	256	256	256	256
Conv layer 3-Pool3	256	256	256	256	256	256
Conv layer 4-Pool4	256	256	512	512	512	512
Conv layer 5-Pool5	512	512	512	512	-	-
Conv layer 6-Pool5	512	512	-	-	-	-
Dense Layer 1 size	1024	1024	1024	1024	1024	1024
Dense Layer 2 size	1024	1024	1024	1024	1024	1024
Output Layer size	10	10	10	10	10	10
Accuracy with all pooling layers	**-**	**-**	**79.96%**	**79.49%**	**80.40%**	**79.72%**
(from experiment 2)			**(ii)**			
Accuracy on removing 1st pooling layer	**83.06%**	**81.72**	**84.10%**	**81.27%**	**83.07%**	**81.11%**
			(iii)			
Accuracy on removing every alternate pooling layer	**84.21%**	**81.04%**	**82.81%**	**79.19%**	**81.55%**	**76.34%**
	(i)		**(iv)**			

- The higher the number of conv layers, the better the accuracy(i). As shown above, network with 6 layers perform better than 5 and 4 layers however it increases system complexity. Since the accuracy was comparable, networks with 4 and 5 layers were chosen for further analysis and 6 layer architecture was dropped.
- Accuracy improves upon dropping the 1st pooling layer (the one right after 1st convolution layer) as compared to a network where every convolution layer is

followed by a pooling layer (ii-iii). This is an important result and hence this idea of dropping first pooling layer was employed in all further architectures.

- Accuracy on removing first pooling layer is better than accuracy on removing every alternate pooling layer. This was an important result and *all further models were designed with pooling starting after the second conv layer rather than 1st*. (iii-iv)
- Accuracy improves to 83%-84%

D. Varying Pooling layers (Without dropout)

In order to confirm the above result of improvement in accuracy upon dropping of the first max pool layer, the same experiment is repeated on the network architectures without including dropout. The set of experiments are summarized below.

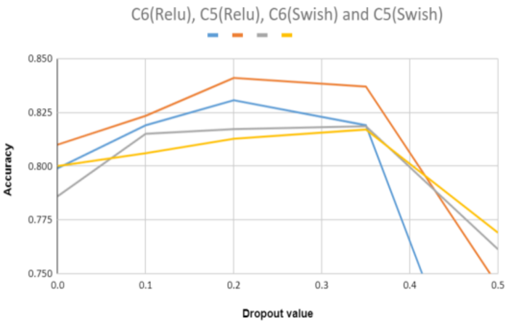
Experiment number 4	Effect of removing MaxPool1 layer (without dropout)			
Number of Conv layers	5	5	4	4
Filter size	3x3	3x3	3x3	3x3
Activation Function	Relu	swish	Relu	swish
Conv layer 1 – No Pool1	128	128	128	128
Conv layer 2-Pool2	256	256	256	256
Conv layer 3-Pool3	256	256	256	256
Conv layer 4-Pool4	512	512	512	512
Conv layer 5-Pool5	512	512	-	-
Dense Layer 1 size	1024	1024	1024	1024
Dense Layer 2 size	1024	1024	1024	1024
Output layer size	10	10	10	10
Accuracy with all pooling layers (from experiment 1)	**74%**	**75%**	**74.36%**	**74.27%**
Test_Accuracy (with no Pool1)	**80.77%**	**78.88%**	**79.73%**	**78.82%**

- Accuracy improves upon dropping the 1st pooling layer (the one right after 1st convolution layer) as compared to a network where every convolution layer is followed by a pooling layer even on architectures without employing dropout. This confirms our findings and hence all further models were designed with pooling starting after the second conv layer rather than the 1st.

E. Varying dropout rates

Architectures with 5 and 6 convolutional layers are employed to study different dropout values from 0 to 0.5 Learning rate is set to 0.0005. The set of experiments are summarized below.

Dropout value	C6(Relu)	C5(Relu)	C6(Swish)	C5(Swish)
0	79.90%	81%	78.60%	80%
0.1	81.90%	82.34%	81.50%	80.60%
0.2	**83.06%**	**84.10%**	**81.72%**	**81.27%**
0.35	81.90%	83.70%	81.85%	81.70%
0.5	65.70%	74.44%	76.13%	76.90%



- Accuracy improves on small dropout values as compared to no dropout.
- Accuracy goes down for higher values of dropouts like 0.5.
- An optimum value of 0.2 gives best accuracy in this case so this value is chosen for final design.

F. Varying learning rates

Architectures with 5 and 6 convolutional layers are employed to study two learning rate values of 0.0005 and 0.0001. Dropout value of 0.2 in introduced after every pooling and both dense layers. The set of experiments are summarized below.

Learning rates	C6(Relu)	C5(Relu)	C6(Swish)	C5(Swish)
0.0005	83.06%	84%	80.59	81%
0.0001	**84.60%**	**84.92%**	**81.72%**	**82.22%**

- Accuracy improves on all architectures as learning rate is changed from 0.0005 to 0.0001 hence 0.0001 is chosen as learning rate for the final architecture.

G. Including batch normalization

Training of deeper neural networks is an extremely time consuming process because the distribution of each layer's inputs change during the training while the parameters of the previous layer change. This makes it hard to train networks.

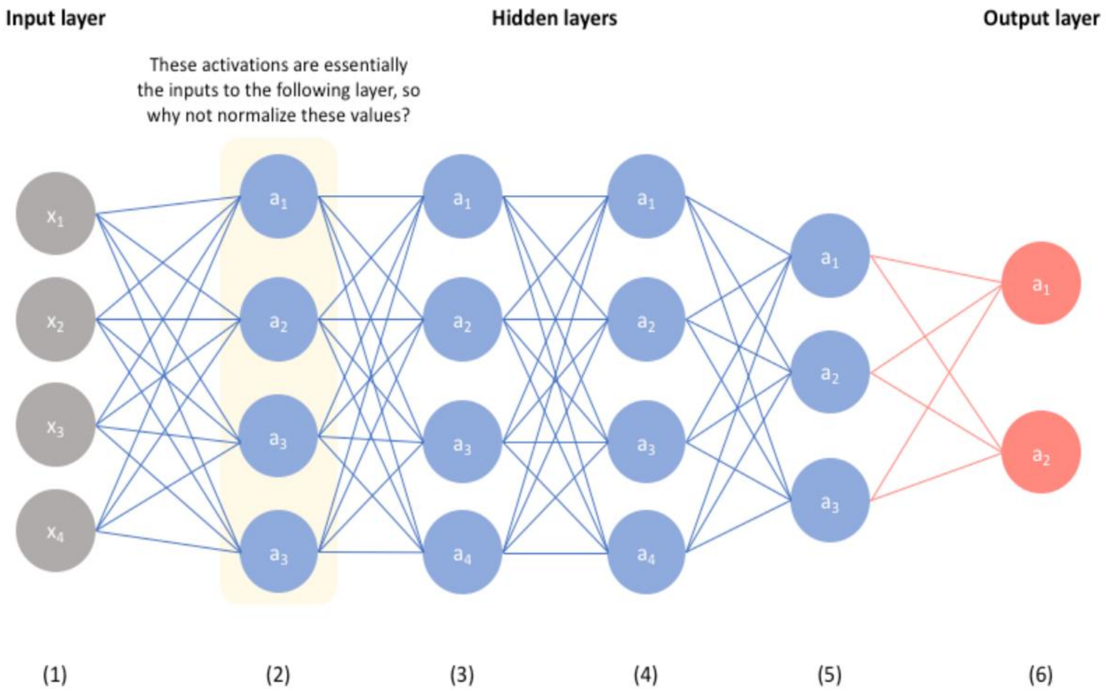

Input layer
Hidden layers
Output layer
These activations are essentially the inputs to the following layer, so why not normalize these values?
x1
x2
x3
x4
a1
a2
a3
a3
a1
a2
a3
a4
a1
a2
a3
a4
a1
a2
a3
a1
a2
(1)
(2)
(3)
(4)
(5)
(6)

Figure 2: Concept of batch normalization[7]

- Batch Normalization is a technique to normalize each layer inputs and it accelerates the training of very deep neural networks [6] Just like normalizing the input data improves the convergence of our network by ensuring the model trains to learn parameters in first layer, batch normalization ensures the activations from previous layers as inputs are normalized. The model learns the parameters in next layers better and the model training becomes smoother.

- Architectures with 5 and 6 convolutional layers are employed to study the effect of including batch normalization layers after all the pooling and both dense layers. Learning rate values is set to 0.0001. Dropout value of 0.2 in introduced after every pooling and both dense layers. The set of experiments are summarized below.

Batch Normalization	C6(Relu)	C5(Relu)	C6(Swish)	C5(Swish)
No	84.60%	84.92%	80.59%	82%
Yes	**86.08%**	**86.79%**	**86.28%**	**86.14%**

- Implementation of batch normalization helps in training with a lesser number of epochs while achieving an improved accuracy compared to model with no batch normalization. Validation accuracy increases from 84.82 to 86.79%. Hence batch normalization was implemented in the final architecture.

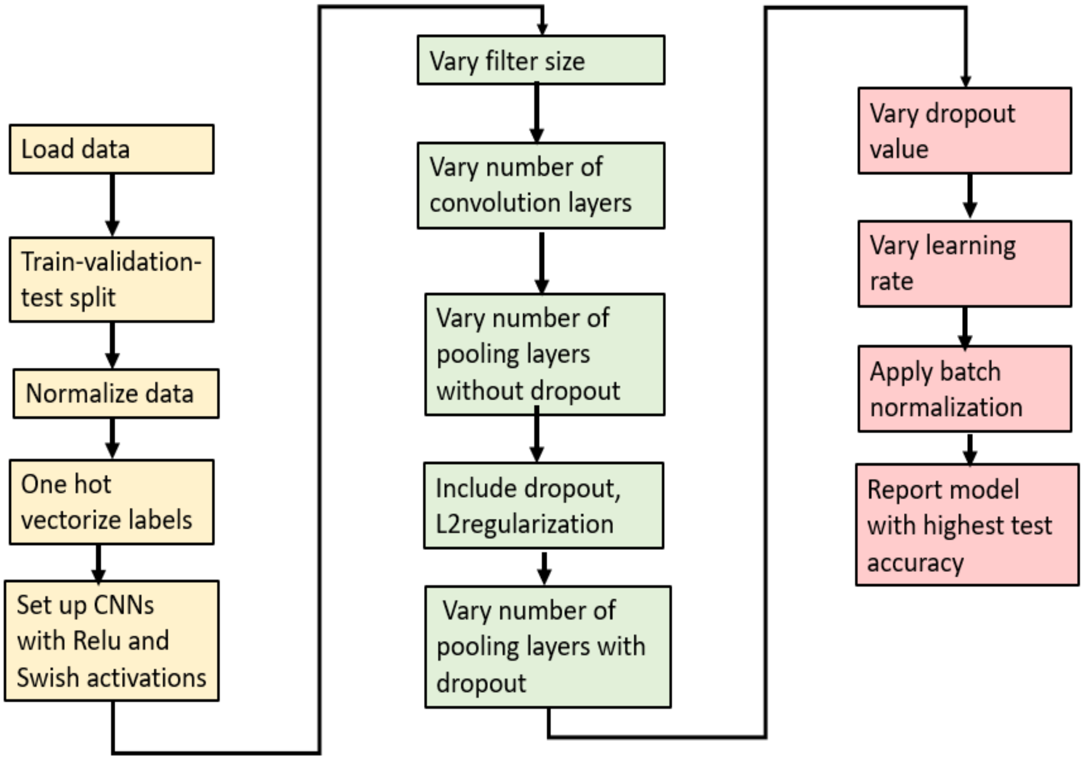

Load data
Train-validation-test split
Normalize data
One hot vectorize labels
Set up CNNs with Relu and Swish activations
Vary filter size
Vary number of convolution layers
Vary number of pooling layers without dropout
Include dropout, L2regularization
Vary number of pooling layers with dropout
Vary dropout value
Vary learning rate
Apply batch normalization
Report model with highest test accuracy

Figure 3: Sequence of steps to obtain parameters of the final network

Final Network Architecture

After completing all the experiments, the model with highest test accuracy is chosen to implement the final architecture shown below.

Input size (32*32*3) Padding = same

Number of kernels in 1st convolution layer	128
Size of kernel in 1st convolution layer	3*3
Stride and activation of 1st conv layer	1*1 Relu
Size of 128 maps	32*32 (32+2-3+1)
Number of kernels in 2ndconvolution layer	256
Size of kernel in 2nd convolution layer	3*3
Stride and activation of 2nd conv layer	1*1 Relu
Size of 256 maps	32*32 (32+2-3+1)
Size and stride of 2nd pooling layer	2*2 2
Size of 256 pooled maps	16*16 (32/2)
Number of kernels in 3rd convolution layer	256
Size of kernel in 3rd convolution layer	3*3
Stride and activation of 3rd conv layer	1*1 Relu
Size of 256 maps	16*16 (16+2-3+1)
Size and stride of 3rd pooling layer	2*2 2
Size of 256 pooled maps	8*8 (16/2)
Number of kernels in 4th convolution layer	512
Size of kernel in 4th convolution layer	3*3
Stride and activation of 4th conv layer	1*1 Relu
Size of 512 maps	8*8 (8+2-3+1)
Size and stride of 4th pooling layer	2*2 2
Size of 512 pooled maps	4*4 (8/2)
Number of kernels in 5th convolution layer	512
Size of kernel in 5th convolution layer	3*3
Stride and activation of 5th conv layer	1*1 Relu
Size of 512 maps	4*4 (4+2-3+1)
Size and stride of 5th pooling layer	2*2 2
Size of 512 pooled maps	2*2 (4/2)
Flatten layer	2048 neurons (2*2*512)
Dense layer1 and its activation	1024 Relu/swish
Dense layer2 and its activation	1024 Relu/swish
Dropout value	0.2 (after all pooling and both dense layers)
Batch normalization	after all pooling layers and both dense layers
Output layer	10 neurons
Activation	Softmax
Cost function	Categorical_crossentropy
Optimizer	Adam
Learning rate	0.0001
Weight initializer used for each layer	glorot normal on both dense and output layers

Mini-batch size used	32
Epochs	25
lmda value both for the regularizer at both the dense layers and at the output layer	0.00001
Loss on training data	0.16
Loss on validation data	0.55
Loss on test data	0.56
Accuracy on training data	95.03%
Accuracy on validation data	87.12%
Accuracy on test data	85.21%

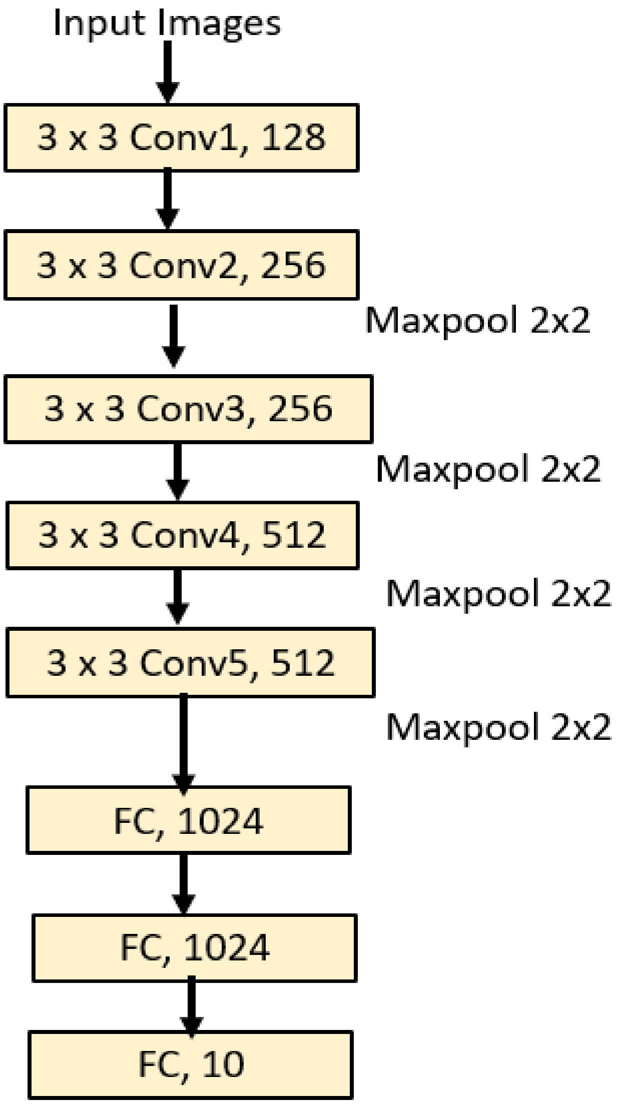
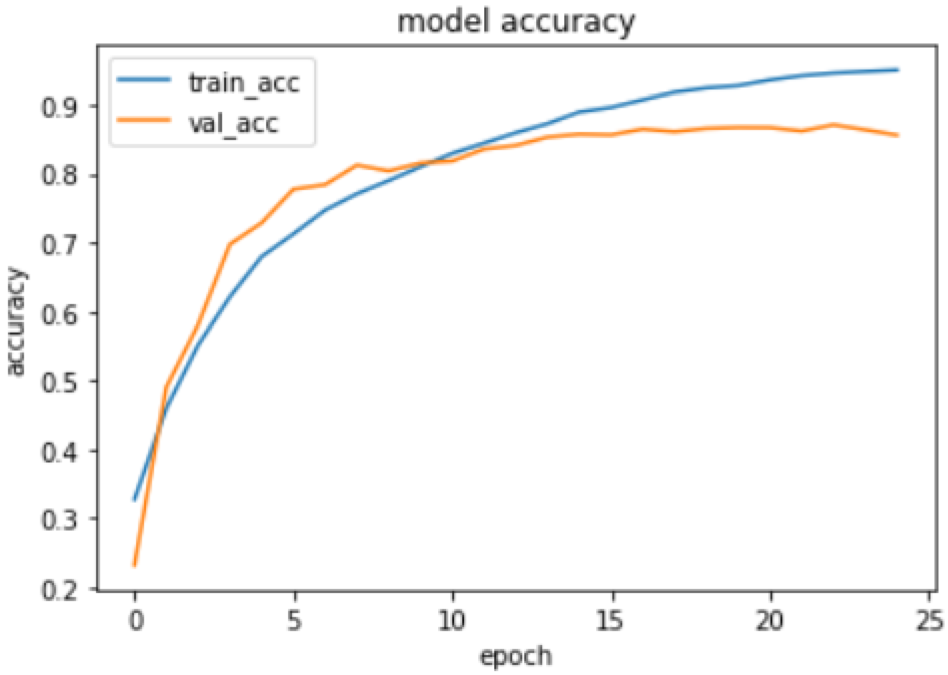
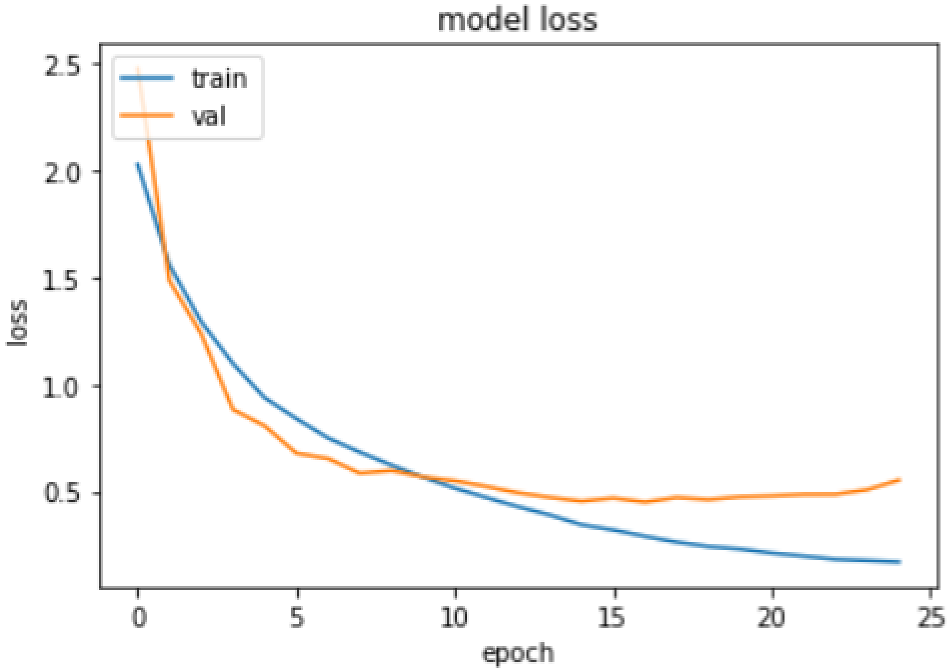


Figure 4 CNN architecture for final model



Accuracy on Training data: 95.02% **Accuracy on Validation data: 87.12%**

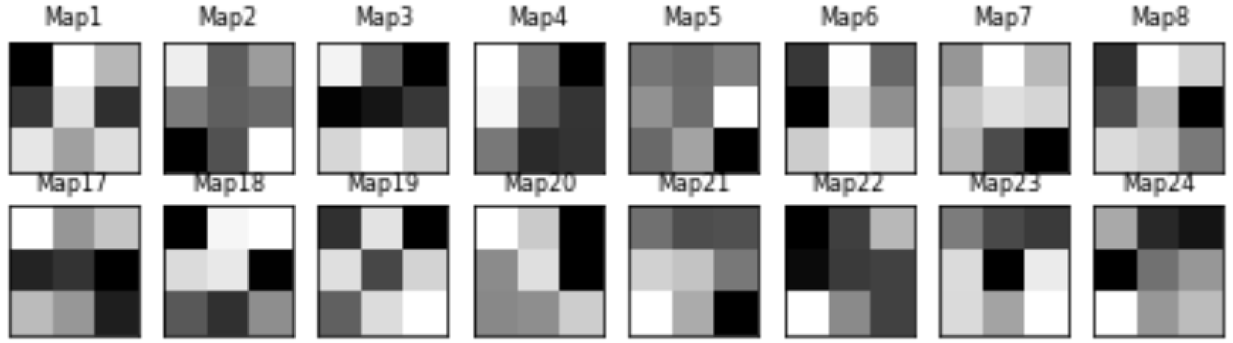


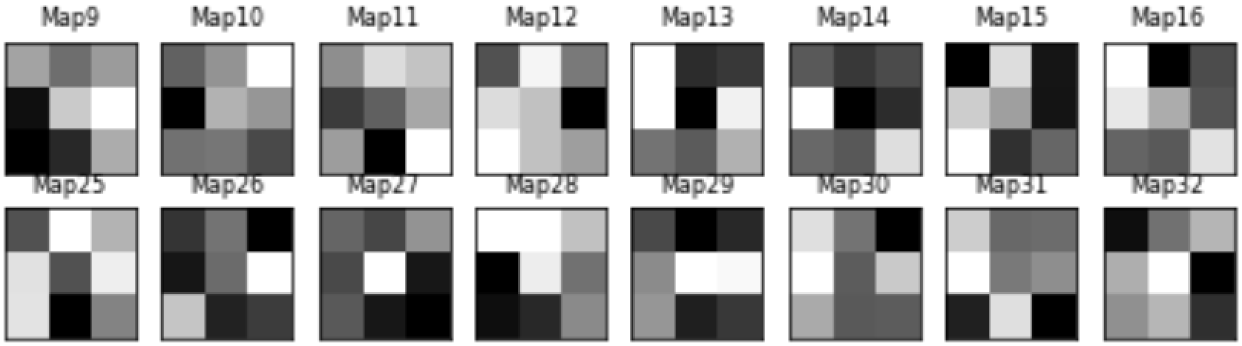
Loss on Training data: 0.16 **Loss on Validation data: 0.44**

Convolution Weights of the First Layer

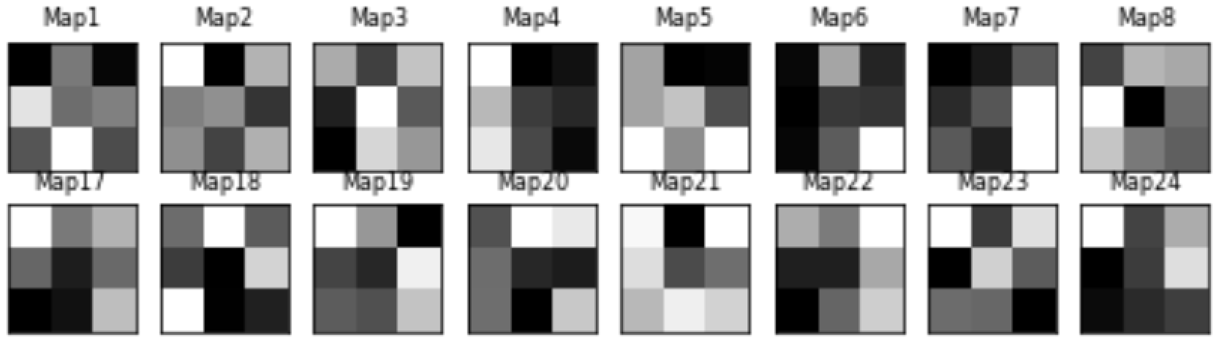
The first layer is a convolution layer with 128 maps. After training we have 128 convolution filters of size 3×3×3 (here mask size is 3x3 and color image is 3 channel RGB). The plots of all 3 channels of these 2-dimensional filters show what feature each one of them is looking for.

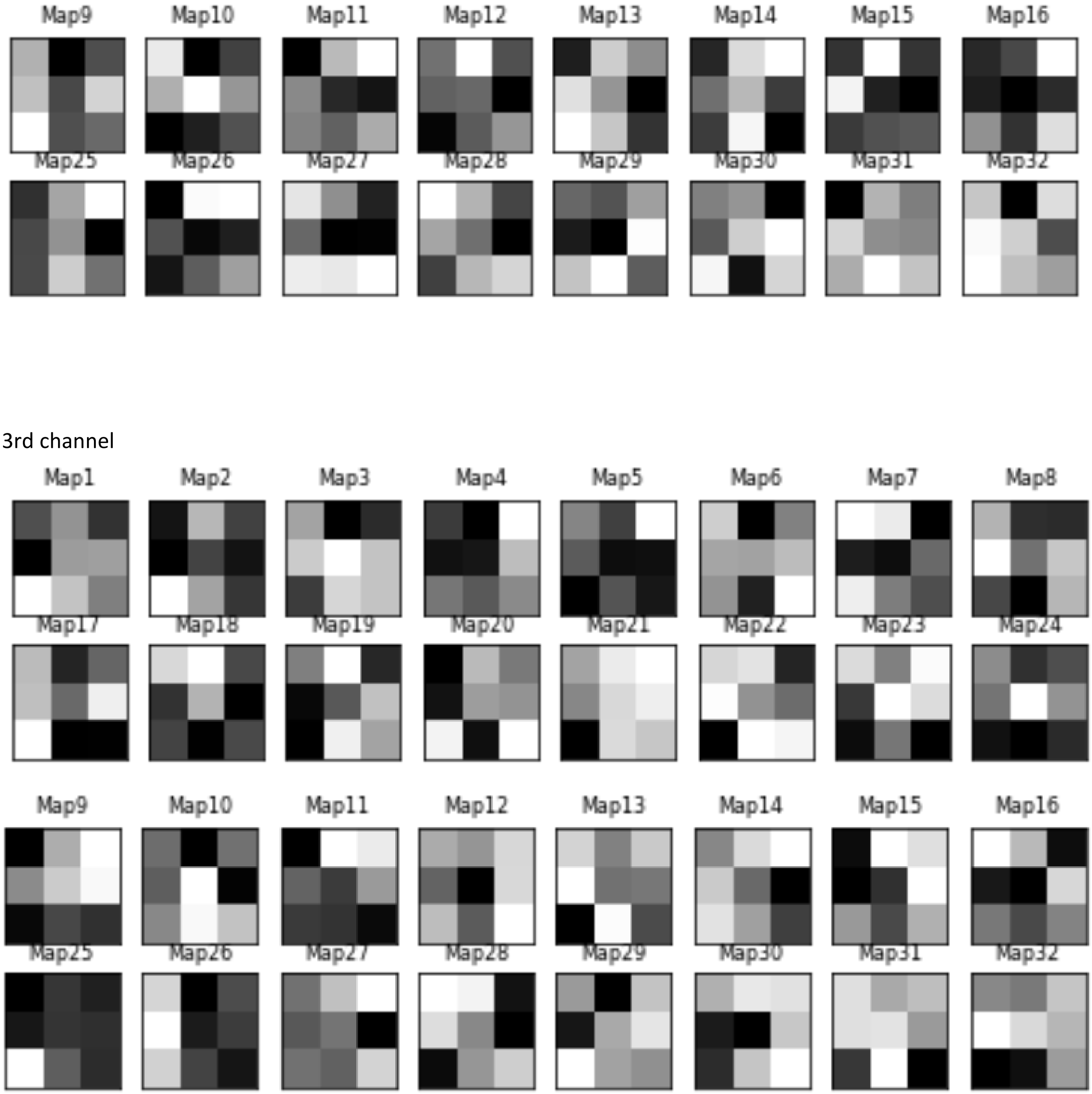
1st channel





2nd channel





3rd channel

Figure 6 Plots of weights (kernels) for all 3 channels for 1st conv layer

Maps of the First Convolution Layer

We choose second image from our data set which is that of a truck. The output plots for each of the first layer neuronal maps is shown below

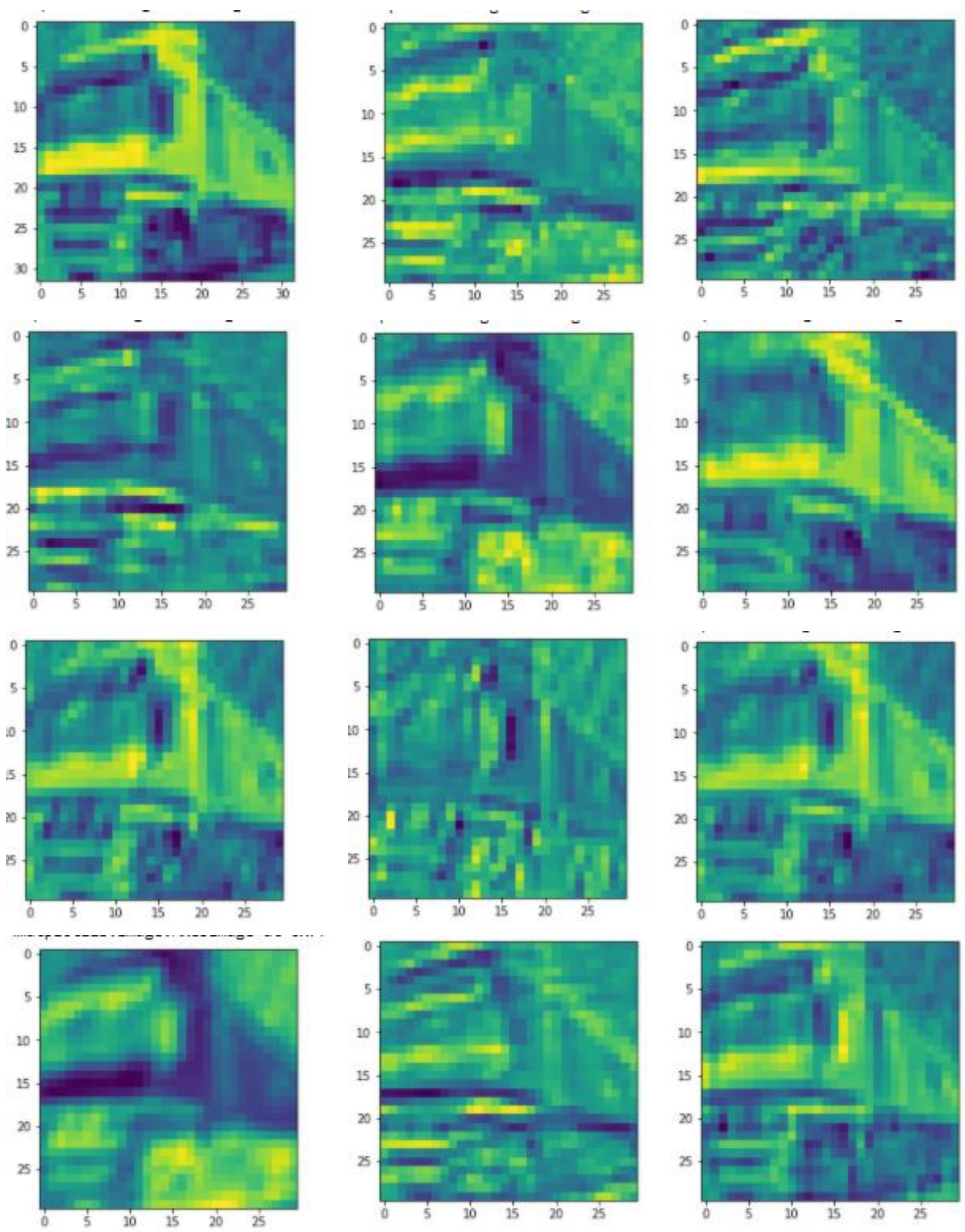


Figure 7 Different convolutions of an image of a truck for 1st channel for 1st conv layer

Results

- Network Performance

The 5 layered convolutional neural network with mask size (3x3 stride of 1), maxpool layers (2x2 stride of 2), employing Relu activation, two fully connected layers with 1024 neurons each, output layer with 10 neurons and softmax activation, dropout (0.2) L2 regularization(0.00001) and batch normalization, batch size 32, epochs 25 gives following results-

Loss on training data	0.16
Loss on validation data	0.55
Loss on test data	0.56
Accuracy on training data	95.03%
Accuracy on validation data	87.12%
Accuracy on test data	85.21%

Learnable filters and pooling layers are used to extract underlying image features. Dropout, regularization and batch normalization are applied to reduce overfitting while ensuring increased accuracies in validation and testing.

- Comparison with literature

Huang et al. achieved an accuracy of 96.54% on this dataset [1]. They employed Dense Convolutional Network (DenseNet) architecture which connects each layer to every other layer in a feed-forward fashion. They used more than 40 number of total layers. A much smaller, faster and simpler CNN model in this project achieves an appreciable test accuracy of **85.21%** on test data in just 25 epochs while using only a total of 5 convolution layers.

Also, this project clearly shows that the recently developed Swish activation lends itself useful in image classification problems to train deep neural networks. Swish performs comparable and, in some cases, outperforms Relu activations and is very simple to implement.

References

[1] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely Connected Convolutional Networks.”

[2] P. Ramachandran, B. Zoph, and Q. V. Le, “Searching for activation functions,” 2017.

[3] “Manning | Deep Learning with Python.” [Online]. Available: https://www.manning.com/books/deep-learning-with-python. [Accessed: 13-Dec-2020].

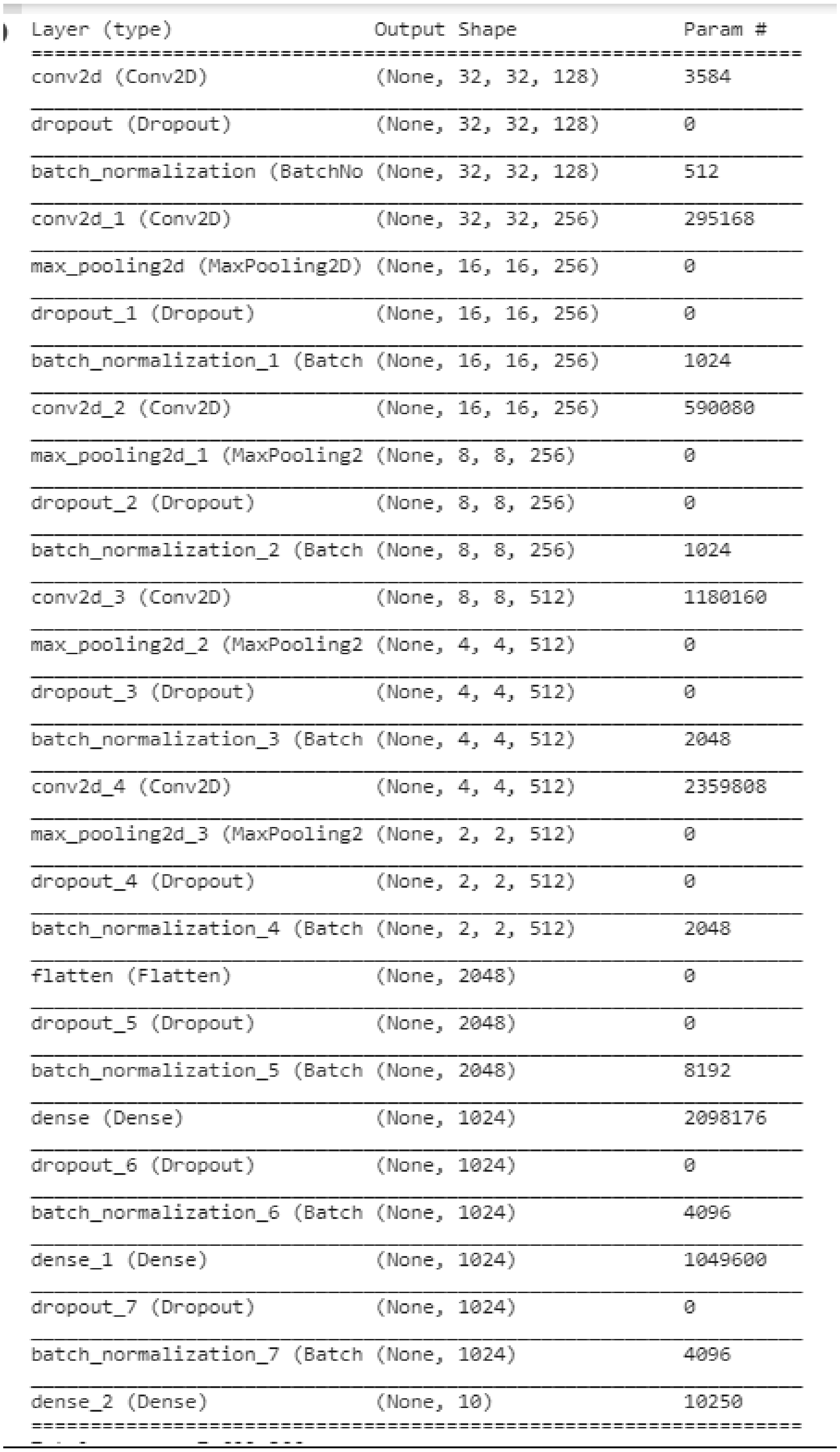
[4] M. A. Nielsen, “Neural Networks and Deep Learning.” Determination Press, 2015.

[5] “Keras Conv2D and Convolutional Layers - PyImageSearch.” [Online]. Available: https://www.pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/. [Accessed: 15-Dec-2020].

[6] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *32nd International Conference on Machine Learning, ICML 2015*, 2015, vol. 1, pp. 448–456.

[7] “Normalizing your data (specifically, input and batch normalization).” [Online]. Available: https://www.jeremyjordan.me/batch-normalization/. [Accessed: 15-Dec-2020].

<u>Appendix</u> Model Summary



```
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 32, 32, 128)       3584      
_________________________________________________________________
dropout (Dropout)            (None, 32, 32, 128)       0         
_________________________________________________________________
batch_normalization (BatchNo (None, 32, 32, 128)       512       
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 256)       295168    
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 256)       0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 256)       0         
_________________________________________________________________
batch_normalization_1 (Batch (None, 16, 16, 256)       1024      
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 256)       590080    
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 256)         0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 8, 8, 256)         0         
_________________________________________________________________
batch_normalization_2 (Batch (None, 8, 8, 256)         1024      
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 8, 8, 512)         1180160   
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 512)         0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 4, 4, 512)         0         
_________________________________________________________________
batch_normalization_3 (Batch (None, 4, 4, 512)         2048      
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 4, 4, 512)         2359808   
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 2, 2, 512)         0         
_________________________________________________________________
dropout_4 (Dropout)          (None, 2, 2, 512)         0         
_________________________________________________________________
batch_normalization_4 (Batch (None, 2, 2, 512)         2048      
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0         
_________________________________________________________________
dropout_5 (Dropout)          (None, 2048)              0         
_________________________________________________________________
batch_normalization_5 (Batch (None, 2048)              8192      
_________________________________________________________________
dense (Dense)                (None, 1024)              2098176   
_________________________________________________________________
dropout_6 (Dropout)          (None, 1024)              0         
_________________________________________________________________
batch_normalization_6 (Batch (None, 1024)              4096      
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              1049600   
_________________________________________________________________
dropout_7 (Dropout)          (None, 1024)              0         
_________________________________________________________________
batch_normalization_7 (Batch (None, 1024)              4096      
_________________________________________________________________
dense_2 (Dense)              (None, 10)                10250     
=================================================================
```