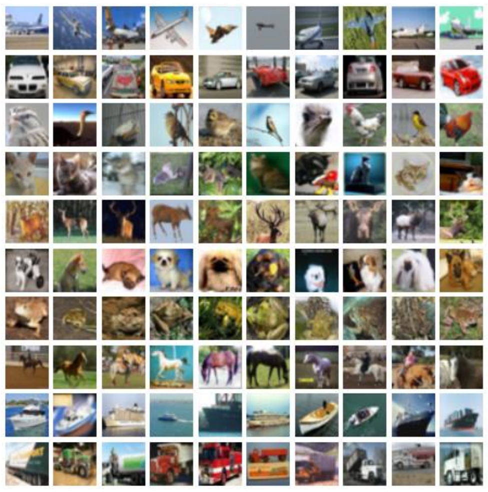
Report for Project-2



COMP 8720 (Topics: Artificial Intelligence)

By:
Jasmin Patel (110095248)
Mahesh Abburi (110095242)
Mohammed Farhan Baluch (110093799)

Professor:
Prof. Robin Gras

University of Windsor
Sept 2022

Participant of the group

Name	Student ID
Mahesh Abburi	110095242
Jasmin Patel	110095248
Mohammed Farhan Baluch	110093799

"As a student of the University of Windsor, we pledge to pursue all endeavors with honor and integrity, and will not tolerate or engage in academic or personal dishonesty. We confirm that we have not received any unauthorized assistance in preparing for or writing this assignment. We acknowledge that a mark of 0 may be assigned for copied work."

TABLE OF CONTENTS

LIST OF FIGURES

Abstract:

In this project, we have performed the image classification task on the CIFAR-10 dataset, where each image belongs to one of the ten distinct classes. The classes are mutually exclusive and are mostly objects and animals. The images are small (32 ×32 pixels), of uniform size and shape, and RGB coloured. We are going to implement a proposed image pre-processing framework to learn and extract the salient features of the images. We further experiment with various models like ANN, the Base model of CNN, Improved CNN with Data argumentation and adding more filters, batch normalization and compare the accuracies of each model and also experimented with a model where we combined all the techniques used in other models. We found out that the model where all the techniques are combined gives the best testing and training accuracies.

Introduction:

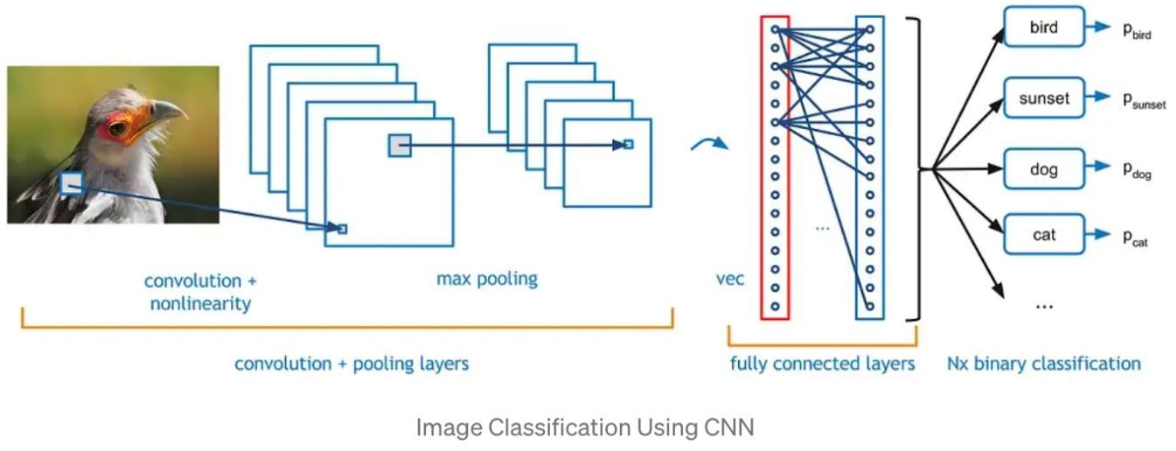


Figure 1 Image classification using CNN

Traditional machine learning methods (such as multilayer perception machines, support vector machines, etc.) mostly use shallow structures to deal with a limited number of samples and computing units. When the target objects have rich meanings, the performance and generalization ability of complex classification problems are obviously insufficient. The convolution neural network (CNN) developed in recent years has been widely used in the field of image processing because it is good at dealing with image classification and recognition problems and has brought great improvement in the accuracy of many machine learning tasks. It has become a powerful and universal deep learning model.

In this project, we are also implementing the CIFAR-10 Image classification using CNN with some improvements to increase the accuracy. our task is to classify the images randomly extracted from the CIFAR-10 dataset. These images can be classified into 10 classes of equal size, where each contains 6,000 images and the classes correspond to mostly objects (such as ships, cars, airplanes, etc.) and animals (cats, dogs, frogs, etc.).

A hundred random sample images from the dataset are shown in Fig. 1 together with their correct classifications. These 10 classes are completely mutually exclusive, e.g., there is no overlapping between automobiles and trucks. Hence, the goal of the project is to label the test images with their correct classes. We will be developing some models and loading the dataset into it and preparing the model for classification.

In our implementation of the image classification, we have experimented with seven different models like ANN, Base model of CNN, Improved CNN and Data Augmentation, Base model + Dropouts, Base model + Dropouts + more filters + Batch Normalization, Base Model-2+ Data Augmentation + Batch Normalization + 700 epochs, Baseline + Increasing Dropout + Data Augmentation + Batch Normalization. Pre-processed the data and loaded it into these models to see the accuracy of each model and made a comparison between all these models. Finally, we have implemented a model with the combination of all the above seven models and tested the accuracy of both training and testing. The accuracy of the combined model is higher than the other models which have 99.6% of training accuracy and almost 89% of testing accuracy.

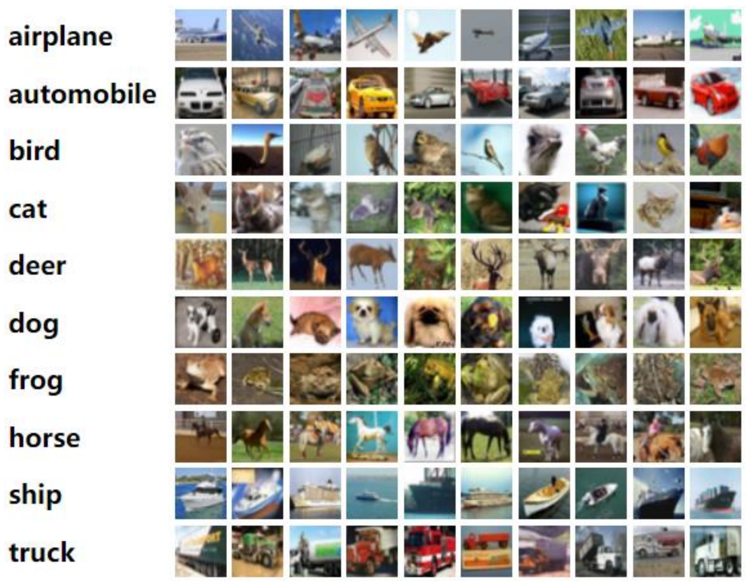


Figure 2 CIFAR Dataset

The CIFAR-10 dataset (Canadian Institute For Advanced Research) is a set of images used to train machine learning and computer vision algorithms.

It is one of the most common datasets used in machine learning research.

The CIFAR-10 dataset comprises 60,000 32x32 colour pictures divided into ten categories. Airplanes, vehicles, birds, cats, deer, dogs, frogs, horses, ships, and trucks are represented by the ten various classes. Each class has 6,000 photos.

Computer algorithms that detect things in photographs often learn by example. CIFAR-10 is a collection of photographs aimed at teaching a computer to detect items. Because the pictures in CIFAR-10 are low-resolution (32x32), researchers may quickly test alternative methods to determine what works.

Various Research questions were arising while proceeding with the problem statement. Some od them are listed below and other are taken into consideration with the discussion part of report.

1. How to improve the accuracy of the CNN model for image classification?
2. What happens if different inputs are introduced in layers?
3. What happens if a new layer is introduced?
4. Can accuracy be improved if more Filters, Data argumentation and batch normalization are introduced in the CNN model?

Hardware Requirements:

Instead Considering the average hardware requirements for the development team, the following resources will be required. The average hardware resources for our project are as follows:

1. RAM: 8 GB
2. SSD: 500 GB
3. 8 core CPU

Software Requirements:

1. IDE – Anaconda's Jupyter Notebook
2. Websites – Kaggle, Goggle collab
3. Programming Language - Python
4. Libraries used – Keras, TensorFlow, NumPy, Matplotlib, Sklearn, Pandas, seaborn, pickle, visualkeras, PIL

State of the art:

Graham formulated a novel approach for CNN – he suggested using a fractional version of max-pooling to increase performance over CNN with integer values for max-pooling.
He implemented the approach on the CIFAR-10 dataset with dropout and affine transformations. He added random shifts to images and used a pseudorandom overlapping pooling with the fractional max-pooling network.
Results for the CIFAR-10 dataset showed an error of 3.47%, i.e., 96.53% accuracy with 100 tests. [1]

Literature Review:

Tobias et al. proposed an approach by replacing any pooling or alternating convolution and just including convolution layers in the pipeline.
They replaced the pooling layer with a standard convolution layer with a stride more significant than 1.
The method resulted in an accuracy of 90.92% without data augmentation when used with a stochastic gradient and dropout along with 0.9M fixed momenta. While it showed an accuracy of 95.59% when applied with considerable data augmentation by replacing 32px images with 126px. [2]

Dmytro et al. propose a novel LSUV initialization method for initializing weights in deep learning by pre-initializing the convolution layer weights with orthonormal matrices.
They implemented FitNet4 architecture with LSUV initialization and 0.9 momenta on the CIFAR-10 dataset with 230 epochs.
Results showed a near state-of-the-art performance of 93.94% accuracy with data augmentation. The proposed LSUV outperforms orthonormal initialization by a smaller margin. [3]

Lee et. Al. proposed a generalized version of pooling in the form of 2 strategies – mixed & gated pooling. They also propose a tree pooling in which the model learns pooling filters and combines them for the best result.
Results from the method tree+gated max-avg pooling displays an error % of 7.62, i.e., an accuracy of 92.38% without data augmentation and only 72 additional parameters instead of hundreds used in other methods. [4]

Snoek et al. attempt to tackle efficient optimization of black-box functions using neural networks as an alternative for Gaussian processes. They use a Bayesian optimization strategy and attempt to create a general approach for optimization.
They demonstrate hyperparameter optimization of deep CNN models using the above approach.
Results show an error % of 6.37%, i.e., 93.63% accuracy for the CIFAR-10 dataset. [5]

He et al. propose a residual network framework by reformulating layers as residual functions concerning the layer inputs.
They implemented the 34 parameter layers residual network architecture on CIFAR-10 with a global average pooling, 10-way fully connected layer, and softmax.
They compared the results by changing the values of the number of layers and parameters for ResNet. Results showed the highest accuracy of 93.57% on 110 layers and 1.7M parameters. [6]

Clevert et al. Propose a new linear activation function – Exponential linear unit(ELU) to optimize the learning speed in deep neural networks. ELUs with smaller inputs saturate to negative values, decreasing the forward propagated variation.
Results show a good accuracy of 93.45% in CNN using the ELU function. [7]

Methods:

We used various techniques from the field of deep learning to experiment with cifar-10 dataset. Our approach to these methods is listed below. Each methods would be represented as M[1] to M[8] in further discussions.

1. **Artificial Neural Networks**
   In beginning, we tried out the dataset with the Artificial Neural Networks. A very basic model with one Flatten layer and three dense layers was used to train the model. Dense layers having 3000, 1000, and 10 units respectively of neurons were taken into consideration. Training and testing features were normalized between 0-1 for the input of the neural networks. Previously we had a range of 0-255, where it represented the pixel value of RGB colour. Also, classes were converted using one hot encoding to match the output from final dense layer consisting of 10 units.

2. **Base model for CNN**
   Firstly, we build a simple baseline Convolutional Neural Network to check the dataset performance. It consisted of two pairs of Conv2D and MaxPool2D layers. Followed by the same flatten and dense layers from the previous ANN model with a change in the number of units. In total now our model is crossing 1 million parameters.

3. **Improved CNN and Data Augmentation**
   After that, we decided to include some data augmentation techniques for improvement of models. We tried to randomly shift images in the range of 0 to 45 degrees, shift images horizontally and vertically with a fraction of 0.2 of total width and height. Additionally, we enabled random horizontal and vertical flips. We also added multiple convolutional layers with activation layer in this method. At last, we also made a change by adding global pooling layer followed by activation layer other than dense.

4. **Base model + Dropouts + Maxpooling**
   Dropout has shown improvements in the performance of neural networks on supervised learning tasks in vision, speech recognition, document classification and computational biology, obtaining state-of-the-art results on many benchmark data sets. We noticed that validation loss is continuously increasing from the first epoch till the last epoch on cifar-10 dataset i.e., our base model is overfitting. So, to counter this by adding a few dropout layers in our model. Now we build our model by dropping 50% of units. Additionally, we added two maxpooling layer of size (14,14) and (5,5) respectively. By adding Max Pooling we reduced the dimensions of the feature maps. Thus, it reduces the number of parameters to learn and the amount of computation performed in the network.

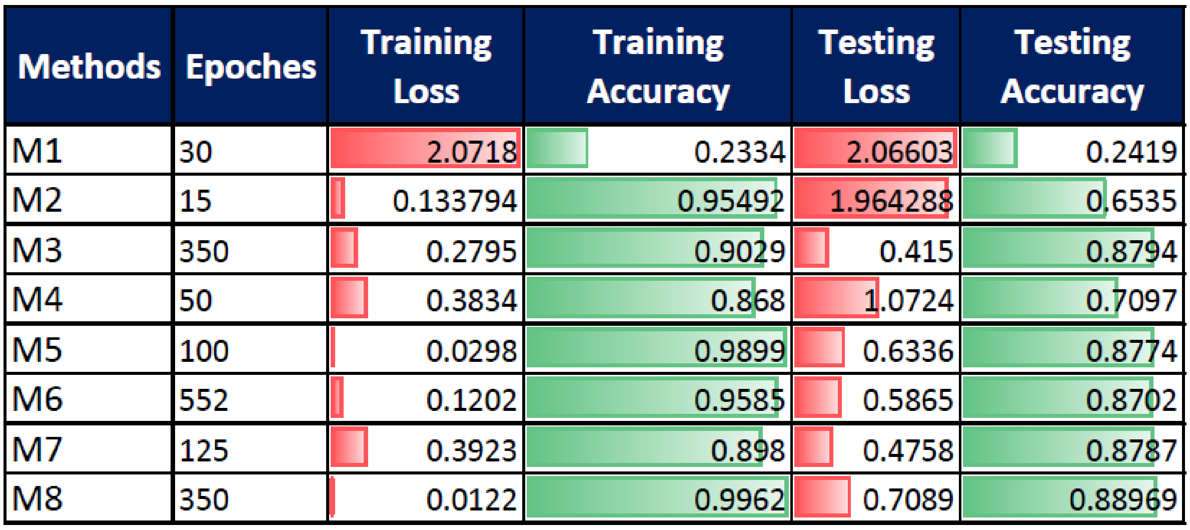
5. **Base model + Dropouts + more filters + Batch Normalization**
   Batch normalization is a technique for training very deep neural networks that normalizes the contributions to a layer for every mini-batch. After adding Batch Normalization layers, It made the instant impact of settling the learning process and drastically decreasing the number of training epochs required to train deep neural networks.

6. **Base Model-2+ Data Augmentation + Batch Normalization + 700 epoches**
   We made a combination of Data Augmentation and Batch Normalization layer after every Convolutional Network. And tried to use lots of epochs to see counter effect to Batch Normalization. To speed up modelling we tried to run overnight on limited free GPUs google collab and Kaggle, we could only run till 553 epochs due to limitation of allowed computes.

7. **Baseline + Increasing Dropout + Data Augmentation + Batch Normalization**
   This model was made with a slight change of previously seen model 5. Only Data augmentation was added as a part of experiment. Also, in addition we introduced weight decays and learning rate of 3.0000e-04

8. **Combining all techniques**
   At last, we combined all the techniques we used so far to test the performance on cifar-10 dataset. At last there were 6 convolutional layers, with the connection of 6 batch normalization layers. And one batch normalization was also connected with one of dense layer. We used max pooling of (16,16), (8,8), and (4,4) respectively. Also in each group there were also dropout layers to control overfitting.

Result and Discussion:



Methods	Epoches	Training Loss	Training Accuracy	Testing Loss	Testing Accuracy
M1	30	2.0718	0.2334	2.06603	0.2419
M2	15	0.133794	0.95492	1.964288	0.6535
M3	350	0.2795	0.9029	0.415	0.8794
M4	50	0.3834	0.868	1.0724	0.7097
M5	100	0.0298	0.9899	0.6336	0.8774
M6	552	0.1202	0.9585	0.5865	0.8702
M7	125	0.3923	0.898	0.4758	0.8787
M8	350	0.0122	0.9962	0.7089	0.88969

Figure 3 Comparison Table

As we know that the class of cifar-10 dataset is equally distributed. i.e., each class has the equal number of samples. Hence accuracy was most preferred matrix to chosen the major for comparison and changes between various methods.

Firstly, using Neural Networks without Convolutional networks was not good idea for the cifar-10 dataset. Even after increasing depth of simple neural network It's not easy to tell if it improves accuracy. It seems that validation accuracy doesn't diverge from the training accuracy. We only achieved ~24% accuracy with [M1].

Our input is an image and by treating it as a flat array we lose the locality on the features. Moreover, having over 3000 inputs adds computational overhead. Convolution Neural

Networks on the other hand are ideal for images (among others) and by learning on the filter parameters and not on the transformation of the image itself we can achieve deeper and more complex architectures with better results. So, all of out next model [M2] to [M8] are based on Convolutional Neural Networks.

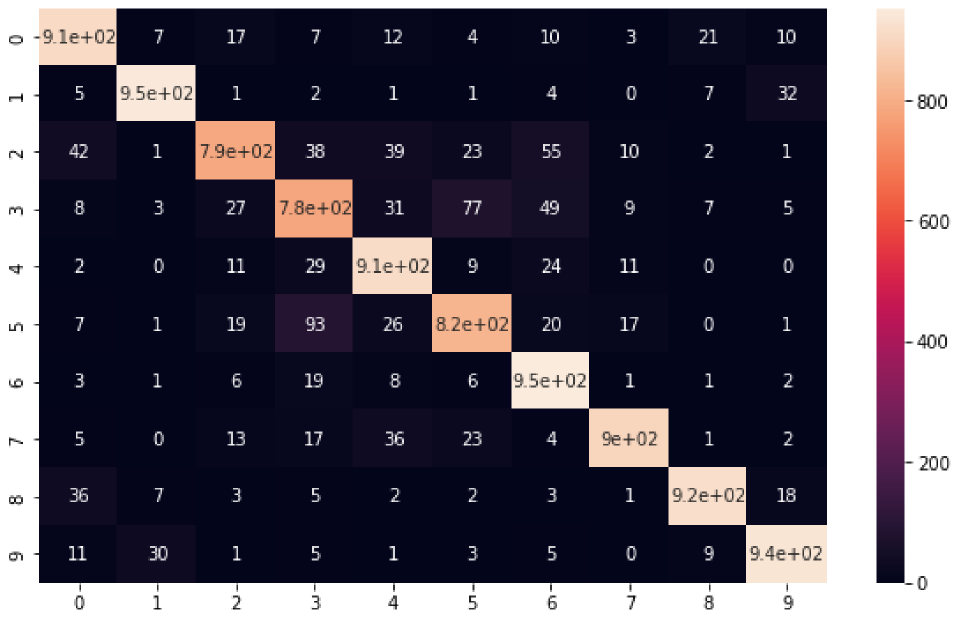
Our simple baseline model of CNN is way better than ANN. It provided a testing accuracy of ~64%. Which is an increase of almost 166%. But to be frank, using this model in real world would not be fruitful, hence for healthy component we introduced other techniques to CNN to increase the output of model. As you can see in the confusion matrix of [M2], classes 0,6-9 are performing well but, still model is making mistake in prediction of class 1-5. Also as you can see in accuracy plot of [M2] validation accuracy is not increasing only train accuracy keeps on increasing. Which suggest that model may be overfitting. Also the validation loss keeps on increasing.

Plots of [M2] were suggesting overfitting with the training data. Hence for further enhancement we made improvement with training data by data augmentation and increased number of training cycles. Because of this changes we reached to 87.9% validation accuracy. Testing loss noted was 1.964, which was still very high. Accuracy and Loss plot of [M3] are also impressive.

To overcome the situation in validation loss as introduced in method-4 description we added dropout layers, but after training with 50 epochs the performance was less this time than expected. It gave 70% accuracy this time. But adding Batch normalization in [M5] it retained the performance of 87% test accuracy once again.

Next thing we tried to check is to check the point where increasing epochs will not improve accuracy. Increasing epochs makes sense only if you have a lot of data in your dataset. As we know we had sufficient datapoints to we set epochs value to 700 and started training in [M6]. But we could only train it up to 552 due to limits of compute system. Training loss was decreased by 8% by this step.

Now for further enhancement we add batch normalization, in an effort to stabilize the learning and perhaps accelerate the learning process. To offset this acceleration, we can increase the regularization by changing the dropout from a fixed pattern to an increasing pattern. In this case, we can see that we achieved a further lift in model performance to about 88% accuracy with the [M7]. Cross entropy loss is decreasing with small rate as training epoch is increased, also result in increase in accuracy.

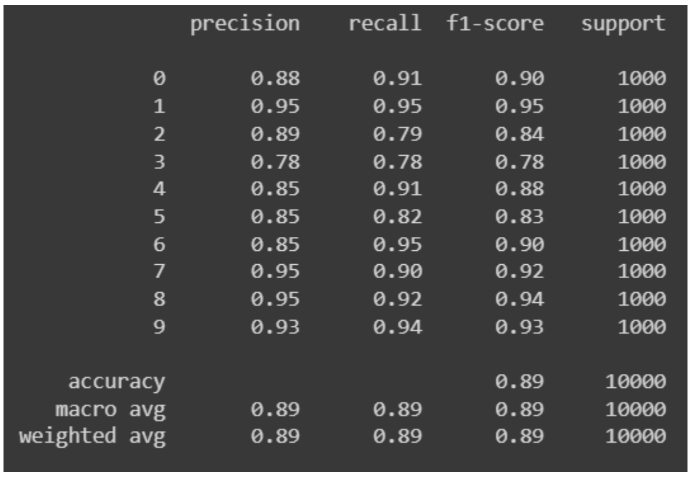


	0	1	2	3	4	5	6	7	8	9
0	9.1e+02	7	17	7	12	4	10	3	21	10
1	5	9.5e+02	1	2	1	1	4	0	7	32
2	42	1	7.9e+02	38	39	23	55	10	2	1
3	8	3	27	7.8e+02	31	77	49	9	7	5
4	2	0	11	29	9.1e+02	9	24	11	0	0
5	7	1	19	93	26	8.2e+02	20	17	0	1
6	3	1	6	19	8	6	9.5e+02	1	1	2
7	5	0	13	17	36	23	4	9e+02	1	2
8	36	7	3	5	2	2	3	1	9.2e+02	18
9	11	30	1	5	1	3	5	0	9	9.4e+02

Figure 4 M8 confusion matrix

The model is currently doing well in terms of learning, and we have strong control over the pace of learning without overfitting. With more regularisation, we might be able to make even more progress. This might be accomplished by employing more aggressive dropout in subsequent layers. It is feasible that adding more weight decay to the model will enhance it. So far, we have not modified the learning algorithm's hyperparameters, such as the learning rate, which is likely the most significant.

We may expect further improvements with adaptive changes to the learning rate. Finally, combining all this techniques in [M8] we made a final model which gave testing accuracy of ~89% . As you can see in plot of [M8] validation accuracy is increasing with a little rate as number if epochs are trained. As you can se in confusion matrix of M8, mostly datapoints are predicted accurately in testing dataset. Model is not precisely predicting class 3 followed by class 5 and class 2. Rest all of them have fantastic f1-scores.



```
              precision    recall  f1-score   support

           0       0.88      0.91      0.90      1000
           1       0.95      0.95      0.95      1000
           2       0.89      0.79      0.84      1000
           3       0.78      0.78      0.78      1000
           4       0.85      0.91      0.88      1000
           5       0.85      0.82      0.83      1000
           6       0.85      0.95      0.90      1000
           7       0.95      0.90      0.92      1000
           8       0.95      0.92      0.94      1000
           9       0.93      0.94      0.93      1000

    accuracy                           0.89     10000
   macro avg       0.89      0.89      0.89     10000
weighted avg       0.89      0.89      0.89     10000
```

Figure 5 M8 classification

Future Works:

Future research should examine if the method is equally resistant to overfitting on different tasks and datasets (such as ImageNet) (such as language modeling). Ensuring that a new test set's data distribution is like the original dataset as feasible is crucial in this situation.

In a broader sense, we see our findings as inspiration for a more in-depth assessment of machine learning research. The overall approach is to propose a new algorithm and assess its performance using existing data. Unfortunately, it is sometimes difficult to determine how generally applicable the gains are. More research is needed to gather new, insightful data to analyze and compare it to the output of existing algorithms to comprehend generalization problems fully. Such investigations adhere to accepted norms of statistically sound research because we already have a sizable body of essentially pre-registered classifiers in open-source repositories.

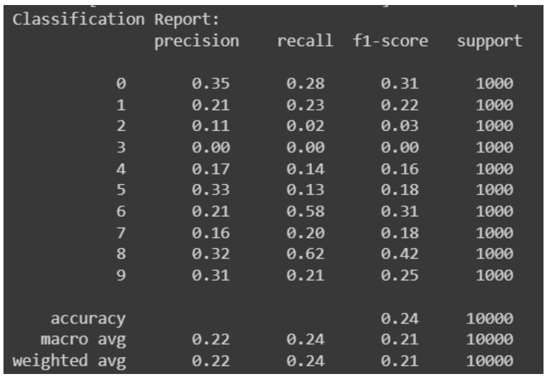
Conclusion:

For this study, we used the CIFAR-10 dataset, where each image is part of one of ten different classes, to conduct the image classification job. The classifications are primarily made up of things and creatures and are mutually exclusive. The pictures are homogeneous in size and shape, tiny (32 by 32 pixels), and RGB-colored. We put the suggested image pre-processing framework into practice to discover and extract the critical aspects of the photographs.
We also tested other models, including ANN, the CNN base model, improved CNN with data argumentation and adding more filters, batch normalization, and dropout, and compared the accuracy of each model. Finally, we tested a model in which we mix all the methods employed in the previous models. We discovered that the model that combines all the strategies has the best training and testing accuracies. The goal was to employ easily accessible methodologies and optimize the model for this dataset to the best of its capacity, even if it is not the greatest in comparison to the state-of-the-art.

References:

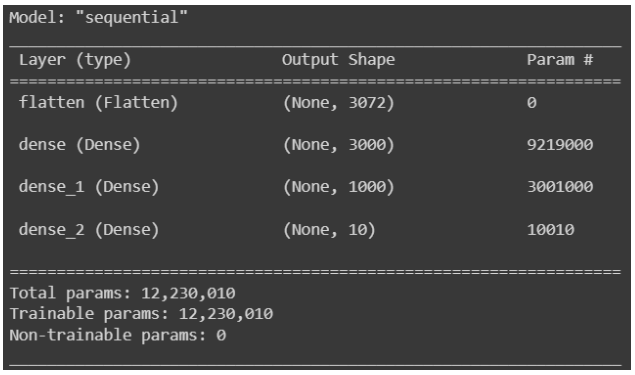
[1] Graham, B. (2014). Fractional max-pooling. arXiv preprint arXiv:1412.6071.
[2] Springenberg, J. T., Dosovitskiy, A., Brox, T., & Riedmiller, M. (2014). Striving for simplicity: The all convolutional net. arXiv preprint arXiv:1412.6806.
[3] Mishkin, D., & Matas, J. (2015). All you need is a good init. arXiv preprint arXiv:1511.06422.
[4] Lee, C. Y., Gallagher, P. W., & Tu, Z. (2016, May). Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. In Artificial intelligence and statistics (pp. 464-472). PMLR.
[5] Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., ... & Adams, R. (2015, June). Scalable bayesian optimization using deep neural networks. In International conference on machine learning (pp. 2171-2180). PMLR.
[6] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).
[7] Clevert, D. A., Unterthiner, T., & Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). arXiv preprint arXiv:1511.07289.

Appendix:



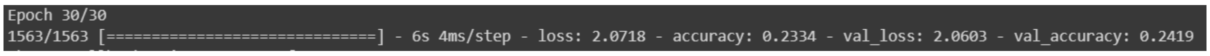
```
Classification Report:
              precision    recall  f1-score   support

           0       0.35      0.28      0.31      1000
           1       0.21      0.23      0.22      1000
           2       0.11      0.02      0.03      1000
           3       0.00      0.00      0.00      1000
           4       0.17      0.14      0.16      1000
           5       0.33      0.13      0.18      1000
           6       0.21      0.58      0.31      1000
           7       0.16      0.20      0.18      1000
           8       0.32      0.62      0.42      1000
           9       0.31      0.21      0.25      1000

    accuracy                           0.24     10000
   macro avg       0.22      0.24      0.21     10000
weighted avg       0.22      0.24      0.21     10000
```

Figure 6 M1 classification report



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 3072)              0

 dense (Dense)               (None, 3000)              9219000

 dense_1 (Dense)             (None, 1000)              3001000

 dense_2 (Dense)             (None, 10)                10010

=================================================================
Total params: 12,230,010
Trainable params: 12,230,010
Non-trainable params: 0
_________________________________________________________________
```

Figure 7 M1 summary



```
Epoch 30/30
1563/1563 [==============================] - 6s 4ms/step - loss: 2.0718 - accuracy: 0.2334 - val_loss: 2.0603 - val_accuracy: 0.2419
```

Figure 8 M1 final epoch

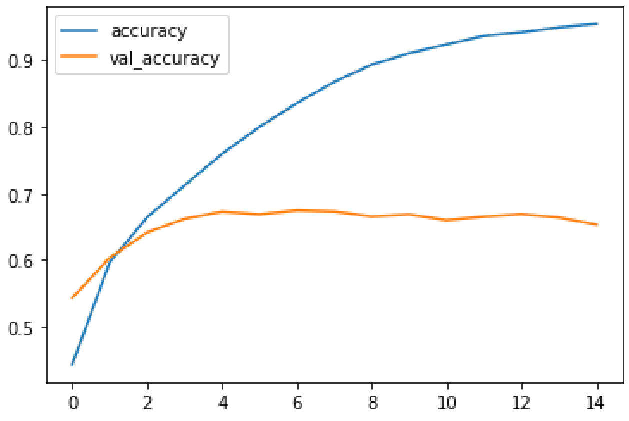
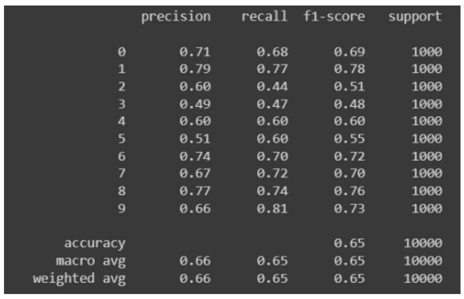
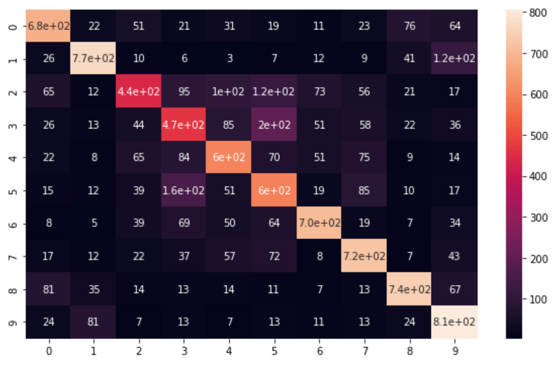


Figure 9 M2 accuracy plot



```
              precision    recall  f1-score   support

           0       0.71      0.68      0.69      1000
           1       0.79      0.77      0.78      1000
           2       0.60      0.44      0.51      1000
           3       0.49      0.47      0.48      1000
           4       0.60      0.60      0.60      1000
           5       0.51      0.60      0.55      1000
           6       0.74      0.70      0.72      1000
           7       0.67      0.72      0.70      1000
           8       0.77      0.74      0.76      1000
           9       0.66      0.81      0.73      1000

    accuracy                           0.65     10000
   macro avg       0.66      0.65      0.65     10000
weighted avg       0.66      0.65      0.65     10000
```

Figure 10 M2 classification report



	0	1	2	3	4	5	6	7	8	9
0	6.8e+02	22	51	21	31	19	11	23	76	64
1	26	7.7e+02	10	6	3	7	12	9	41	1.2e+02
2	65	12	4.4e+02	95	1e+02	1.2e+02	73	56	21	17
3	26	13	44	4.7e+02	85	2e+02	51	58	22	36
4	22	8	65	84	6e+02	70	51	75	9	14
5	15	12	39	1.6e+02	51	6e+02	19	85	10	17
6	8	5	39	69	50	64	7.0e+02	19	7	34
7	17	12	22	37	57	72	8	7.2e+02	7	43
8	81	35	14	13	14	11	7	13	7.4e+02	67
9	24	81	7	13	7	13	11	13	24	8.1e+02

Figure 11 M2 confusion metrix

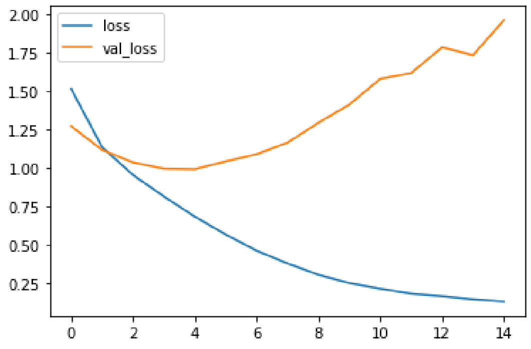
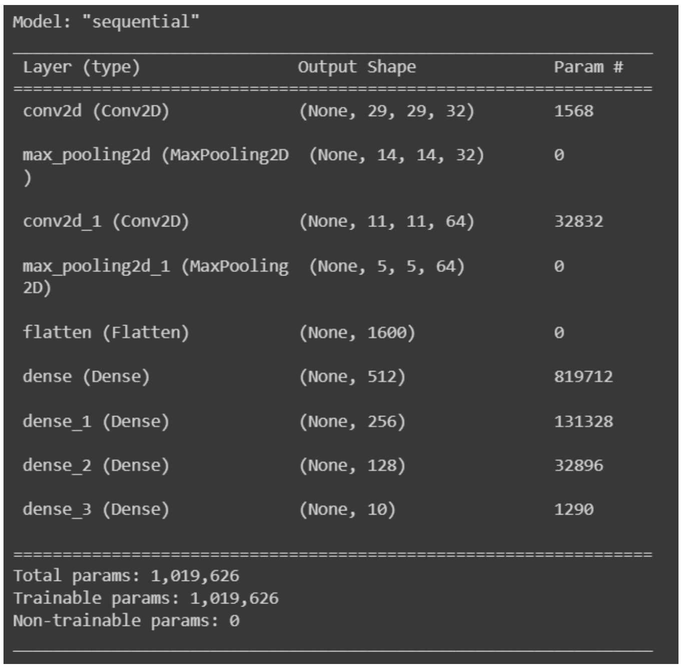
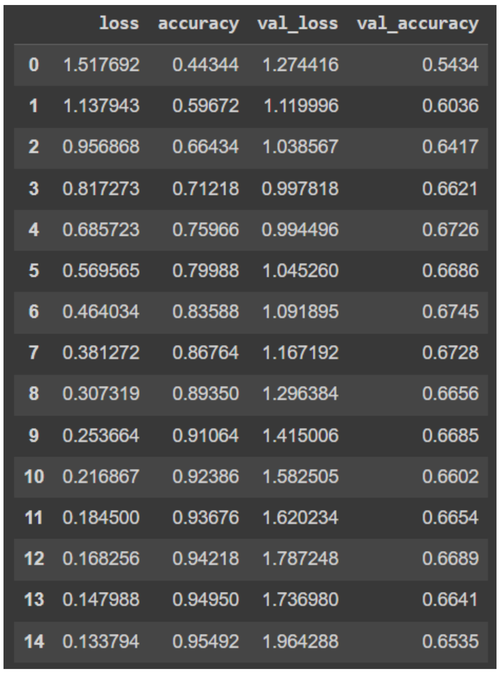


Figure 12 M2 loss plot



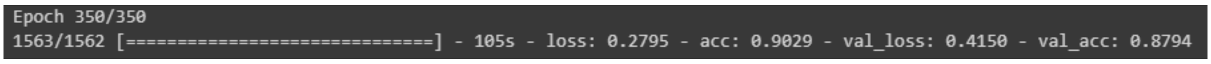
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 29, 29, 32)        1568

 max_pooling2d (MaxPooling2D  (None, 14, 14, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        32832

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 flatten (Flatten)           (None, 1600)              0

 dense (Dense)               (None, 512)               819712

 dense_1 (Dense)             (None, 256)               131328

 dense_2 (Dense)             (None, 128)               32896

 dense_3 (Dense)             (None, 10)                1290

=================================================================
Total params: 1,019,626
Trainable params: 1,019,626
Non-trainable params: 0
_________________________________________________________________
```

Figure 13 M2 summary



	loss	accuracy	val_loss	val_accuracy
0	1.517692	0.44344	1.274416	0.5434
1	1.137943	0.59672	1.119996	0.6036
2	0.956868	0.66434	1.038567	0.6417
3	0.817273	0.71218	0.997818	0.6621
4	0.685723	0.75966	0.994496	0.6726
5	0.569565	0.79988	1.045260	0.6686
6	0.464034	0.83588	1.091895	0.6745
7	0.381272	0.86764	1.167192	0.6728
8	0.307319	0.89350	1.296384	0.6656
9	0.253664	0.91064	1.415006	0.6685
10	0.216867	0.92386	1.582505	0.6602
11	0.184500	0.93676	1.620234	0.6654
12	0.168256	0.94218	1.787248	0.6689
13	0.147988	0.94950	1.736980	0.6641
14	0.133794	0.95492	1.964288	0.6535

Figure 14 M2 training history



```
Epoch 350/350
1563/1562 [============================] - 105s - loss: 0.2795 - acc: 0.9029 - val_loss: 0.4150 - val_acc: 0.8794
```

Figure 15 M3 final epoch

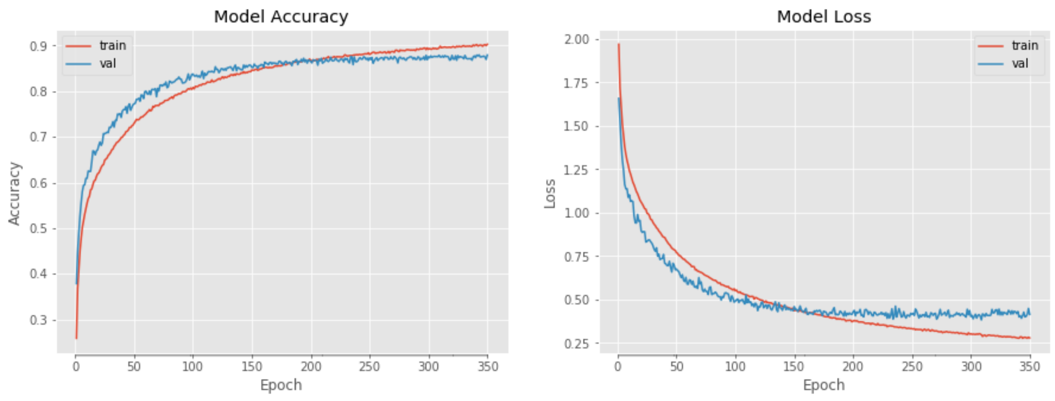
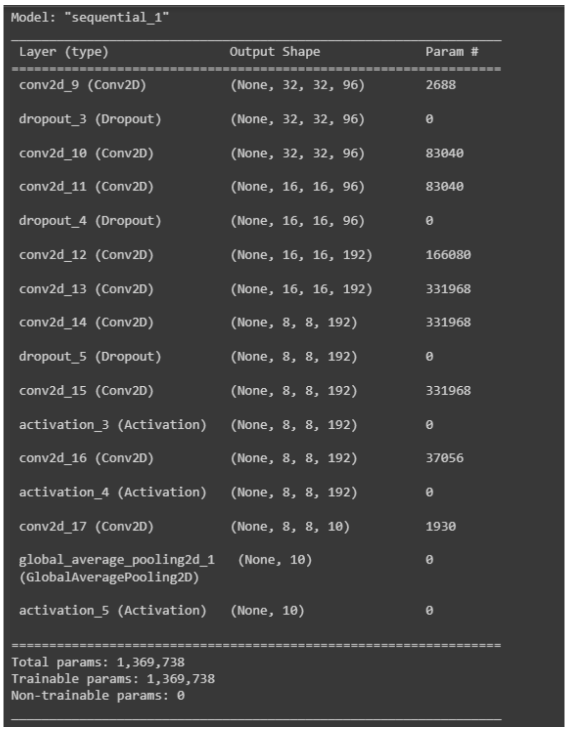
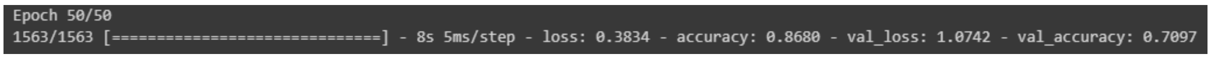


Figure 16 M3 plots



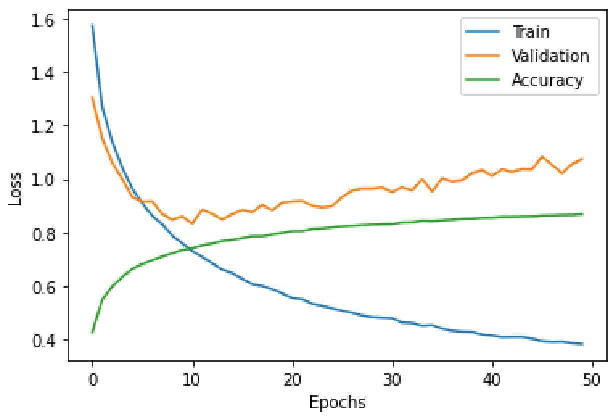
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_9 (Conv2D)           (None, 32, 32, 96)        2688

 dropout_3 (Dropout)         (None, 32, 32, 96)        0

 conv2d_10 (Conv2D)          (None, 32, 32, 96)        83040

 conv2d_11 (Conv2D)          (None, 16, 16, 96)        83040

 dropout_4 (Dropout)         (None, 16, 16, 96)        0

 conv2d_12 (Conv2D)          (None, 16, 16, 192)       166080

 conv2d_13 (Conv2D)          (None, 16, 16, 192)       331968

 conv2d_14 (Conv2D)          (None, 8, 8, 192)         331968

 dropout_5 (Dropout)         (None, 8, 8, 192)         0

 conv2d_15 (Conv2D)          (None, 8, 8, 192)         331968

 activation_3 (Activation)   (None, 8, 8, 192)         0

 conv2d_16 (Conv2D)          (None, 8, 8, 192)         37056

 activation_4 (Activation)   (None, 8, 8, 192)         0

 conv2d_17 (Conv2D)          (None, 8, 8, 10)          1930

 global_average_pooling2d_1   (None, 10)               0
 (GlobalAveragePooling2D)

 activation_5 (Activation)   (None, 10)                0

=================================================================
Total params: 1,369,738
Trainable params: 1,369,738
Non-trainable params: 0
_________________________________________________________________
```

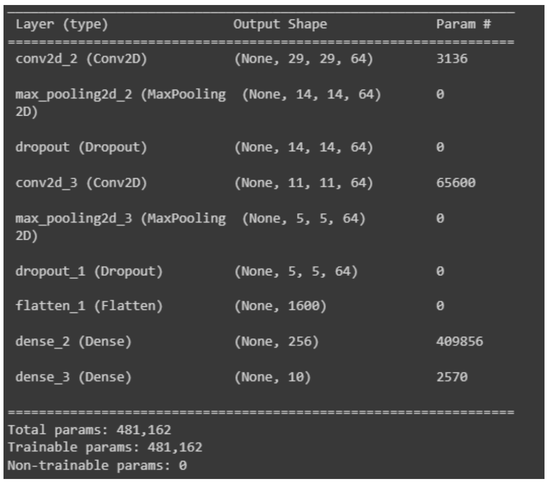
Figure 17 M3 summary



```
Epoch 50/50
1563/1563 [==============================] - 8s 5ms/step - loss: 0.3834 - accuracy: 0.8680 - val_loss: 1.0742 - val_accuracy: 0.7097
```

Figure 18 M4 final epoch

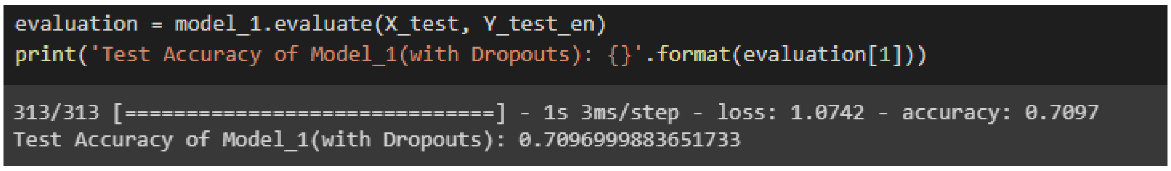




```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 29, 29, 64)        3136

 max_pooling2d_2 (MaxPooling  (None, 14, 14, 64)       0
 2D)

 dropout (Dropout)           (None, 14, 14, 64)        0

 conv2d_3 (Conv2D)           (None, 11, 11, 64)        65600

 max_pooling2d_3 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 dropout_1 (Dropout)         (None, 5, 5, 64)          0

 flatten_1 (Flatten)         (None, 1600)              0

 dense_2 (Dense)             (None, 256)               409856

 dense_3 (Dense)             (None, 10)                2570

=================================================================
Total params: 481,162
Trainable params: 481,162
Non-trainable params: 0
```

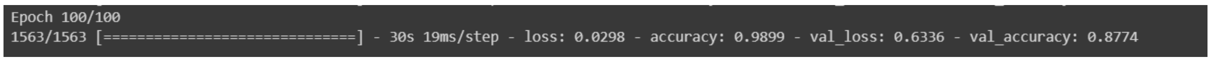
Figure 19 M4 plot

Figure 20 M4 summary



```
evaluation = model_1.evaluate(X_test, Y_test_en)
print('Test Accuracy of Model_1(with Dropouts): {}'.format(evaluation[1]))

313/313 [==============================] - 1s 3ms/step - loss: 1.0742 - accuracy: 0.7097
Test Accuracy of Model_1(with Dropouts): 0.7096999883651733
```

Figure 21 M4 test accuracy



```
Epoch 100/100
1563/1563 [==============================] - 30s 19ms/step - loss: 0.0298 - accuracy: 0.9899 - val_loss: 0.6336 - val_accuracy: 0.8774
```

Figure 22 M5 final epoch

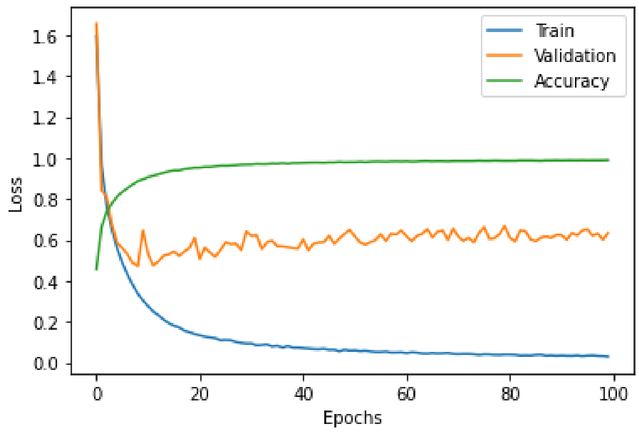
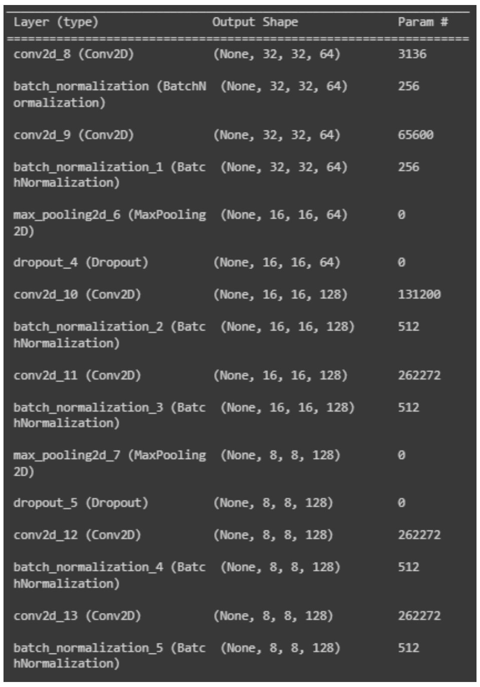
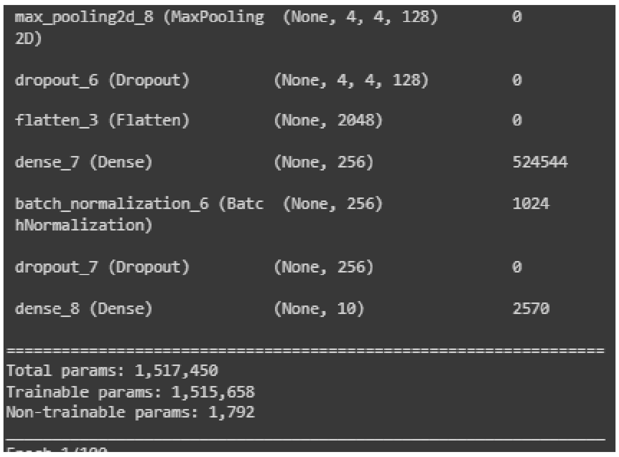


Figure 23 M5 plot



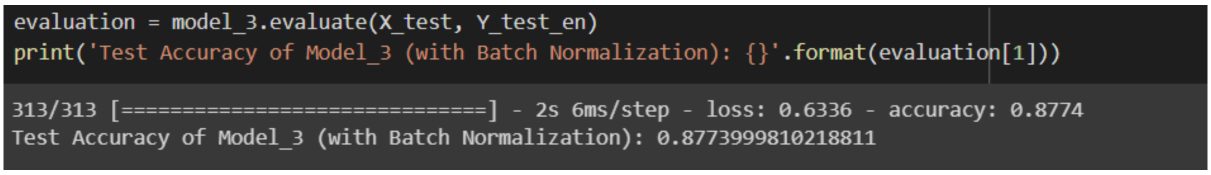
```
Layer (type)                Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)           (None, 32, 32, 64)        3136

batch_normalization (BatchN  (None, 32, 32, 64)       256
ormalization)

conv2d_9 (Conv2D)           (None, 32, 32, 64)        65600

batch_normalization_1 (Batc  (None, 32, 32, 64)       256
hNormalization)

max_pooling2d_6 (MaxPooling  (None, 16, 16, 64)       0
2D)

dropout_4 (Dropout)         (None, 16, 16, 64)        0

conv2d_10 (Conv2D)          (None, 16, 16, 128)       131200

batch_normalization_2 (Batc  (None, 16, 16, 128)      512
hNormalization)

conv2d_11 (Conv2D)          (None, 16, 16, 128)       262272

batch_normalization_3 (Batc  (None, 16, 16, 128)      512
hNormalization)

max_pooling2d_7 (MaxPooling  (None, 8, 8, 128)        0
2D)

dropout_5 (Dropout)         (None, 8, 8, 128)         0

conv2d_12 (Conv2D)          (None, 8, 8, 128)         262272

batch_normalization_4 (Batc  (None, 8, 8, 128)        512
hNormalization)

conv2d_13 (Conv2D)          (None, 8, 8, 128)         262272

batch_normalization_5 (Batc  (None, 8, 8, 128)        512
hNormalization)
```

Figure 24 M5 summary_1



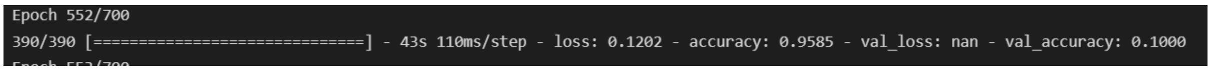
```
max_pooling2d_8 (MaxPooling  (None, 4, 4, 128)        0
2D)

dropout_6 (Dropout)         (None, 4, 4, 128)         0

flatten_3 (Flatten)         (None, 2048)              0

dense_7 (Dense)             (None, 256)               524544

batch_normalization_6 (Batc  (None, 256)              1024
hNormalization)

dropout_7 (Dropout)         (None, 256)               0

dense_8 (Dense)             (None, 10)                2570

=================================================================
Total params: 1,517,450
Trainable params: 1,515,658
Non-trainable params: 1,792
_________________________________________________________________
```

Figure 25 M5 summary_2



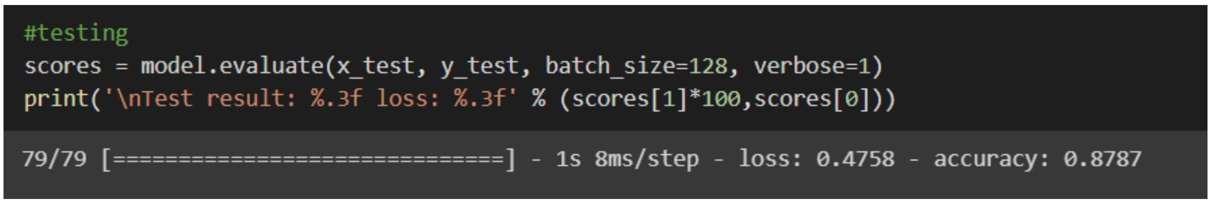
```
evaluation = model_3.evaluate(X_test, Y_test_en)
print('Test Accuracy of Model_3 (with Batch Normalization): {}'.format(evaluation[1]))

313/313 [==============================] - 2s 6ms/step - loss: 0.6336 - accuracy: 0.8774
Test Accuracy of Model_3 (with Batch Normalization): 0.8773999810218811
```

Figure 26 M5 test accuracy



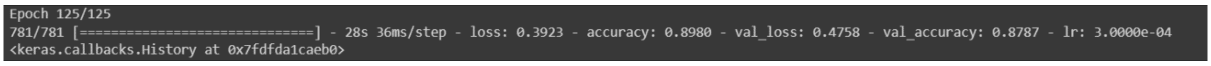
```
Epoch 552/700
390/390 [==============================] - 43s 110ms/step - loss: 0.1202 - accuracy: 0.9585 - val_loss: nan - val_accuracy: 0.1000
```

Figure 27 M6 epoch 552



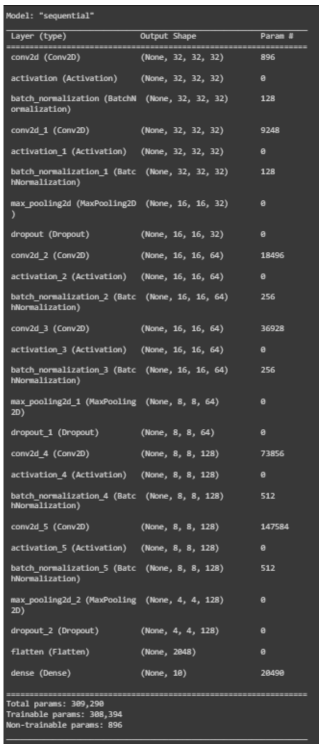
```
#testing
scores = model.evaluate(x_test, y_test, batch_size=128, verbose=1)
print('\nTest result: %.3f loss: %.3f' % (scores[1]*100,scores[0]))

79/79 [==============================] - 1s 8ms/step - loss: 0.4758 - accuracy: 0.8787
```

Figure 28 M7 accuracy test



```
Epoch 125/125
781/781 [==============================] - 28s 36ms/step - loss: 0.3923 - accuracy: 0.8980 - val_loss: 0.4758 - val_accuracy: 0.8787 - lr: 3.0000e-04
<keras.callbacks.History at 0x7fdfda1caeb0>
```

Figure 29 M7 final epoch



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 activation (Activation)     (None, 32, 32, 32)        0

 batch_normalization (BatchN  (None, 32, 32, 32)       128
 ormalization)

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 activation_1 (Activation)   (None, 32, 32, 32)        0

 batch_normalization_1 (Batc  (None, 32, 32, 32)       128
 hNormalization)

 max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0
 )

 dropout (Dropout)           (None, 16, 16, 32)        0

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496

 activation_2 (Activation)   (None, 16, 16, 64)        0

 batch_normalization_2 (Batc  (None, 16, 16, 64)       256
 hNormalization)

 conv2d_3 (Conv2D)           (None, 16, 16, 64)        36928

 activation_3 (Activation)   (None, 16, 16, 64)        0

 batch_normalization_3 (Batc  (None, 16, 16, 64)       256
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 dropout_1 (Dropout)         (None, 8, 8, 64)          0

 conv2d_4 (Conv2D)           (None, 8, 8, 128)         73856

 activation_4 (Activation)   (None, 8, 8, 128)         0

 batch_normalization_4 (Batc  (None, 8, 8, 128)        512
 hNormalization)

 conv2d_5 (Conv2D)           (None, 8, 8, 128)         147584

 activation_5 (Activation)   (None, 8, 8, 128)         0

 batch_normalization_5 (Batc  (None, 8, 8, 128)        512
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 dropout_2 (Dropout)         (None, 4, 4, 128)         0

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 10)                20490

=================================================================
Total params: 309,290
Trainable params: 308,394
Non-trainable params: 896
_________________________________________________________________
```

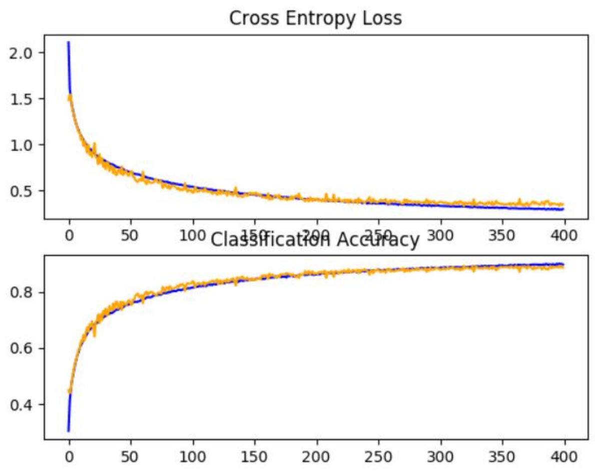


Figure 30 M7 summary

Figure 31 M7 plots

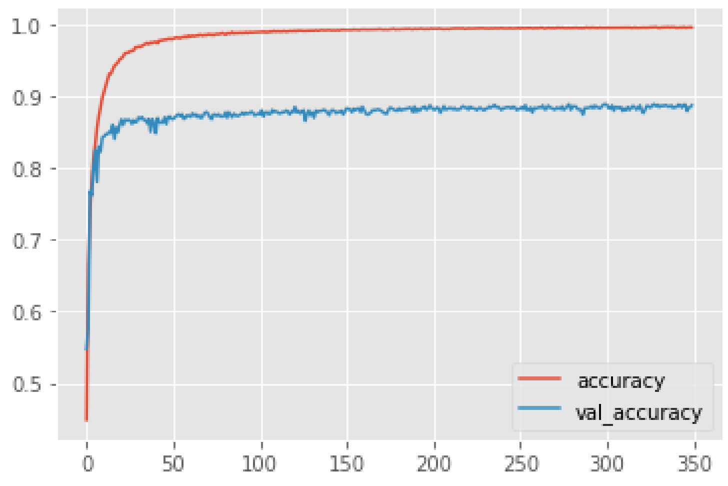
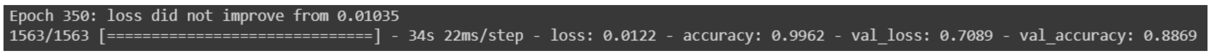


Figure 32 M8 accuracy



```
Epoch 350: loss did not improve from 0.01035
1563/1563 [==============================] - 34s 22ms/step - loss: 0.0122 - accuracy: 0.9962 - val_loss: 0.7089 - val_accuracy: 0.8869
```

Figure 33 M8 final epoch

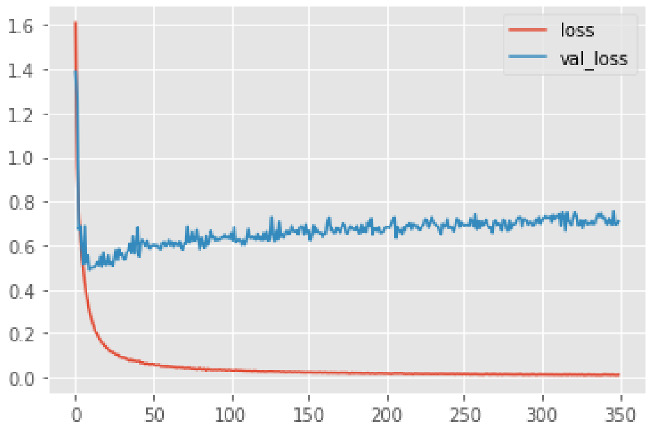
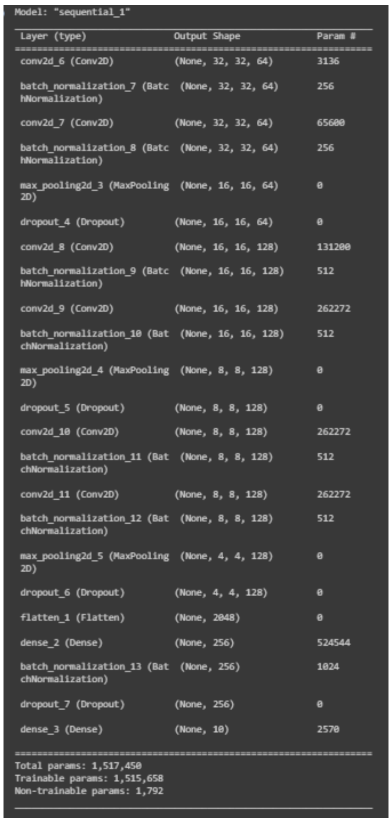


Figure 34 M8 loss



```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 32, 32, 64)        3136

 batch_normalization_7 (Batc  (None, 32, 32, 64)       256
 hNormalization)

 conv2d_7 (Conv2D)           (None, 32, 32, 64)        65600

 batch_normalization_8 (Batc  (None, 32, 32, 64)       256
 hNormalization)

 max_pooling2d_3 (MaxPooling  (None, 16, 16, 64)       0
 2D)

 dropout_4 (Dropout)         (None, 16, 16, 64)        0

 conv2d_8 (Conv2D)           (None, 16, 16, 128)       131200

 batch_normalization_9 (Batc  (None, 16, 16, 128)      512
 hNormalization)

 conv2d_9 (Conv2D)           (None, 16, 16, 128)       262272

 batch_normalization_10 (Bat  (None, 16, 16, 128)      512
 chNormalization)

 max_pooling2d_4 (MaxPooling  (None, 8, 8, 128)        0
 2D)

 dropout_5 (Dropout)         (None, 8, 8, 128)         0

 conv2d_10 (Conv2D)          (None, 8, 8, 128)         262272

 batch_normalization_11 (Bat  (None, 8, 8, 128)        512
 chNormalization)

 conv2d_11 (Conv2D)          (None, 8, 8, 128)         262272

 batch_normalization_12 (Bat  (None, 8, 8, 128)        512
 chNormalization)

 max_pooling2d_5 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 dropout_6 (Dropout)         (None, 4, 4, 128)         0

 flatten_1 (Flatten)         (None, 2048)              0

 dense_2 (Dense)             (None, 256)               524544

 batch_normalization_13 (Bat  (None, 256)              1024
 chNormalization)

 dropout_7 (Dropout)         (None, 256)               0

 dense_3 (Dense)             (None, 10)                2570

=================================================================
Total params: 1,517,450
Trainable params: 1,515,658
Non-trainable params: 1,792
_________________________________________________________________
```

Figure 35 M8 summary