Final Mini-Project B Report

Manash Pratim Barman
Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
mbarman@andrew.cmu.edu

Tejas Mehta
Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
tnmehta@andrew.cmu.edu

1 Introduction

1.1 Problem

We have selected the classification task for the Scalable Methods and Metrics for CIFAR100. Therefore, our goal is - given an input image, the machine learning model would classify/assign it to one of the 100 discrete classes available in the CIFAR 100 dataset.

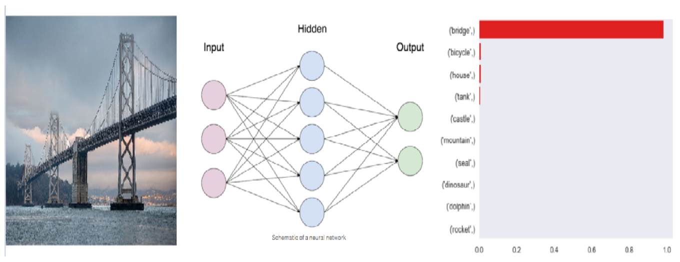
1.2 Motivation

We chose image classification as our ML Task because it would involve an end-to-end machine learning pipeline: data preprocessing, modeling, and evaluating the model on the unseen data. Image classification is also one of the most widely used tasks in Machine Learning, which has many applications in the industry.

1.3 Dataset

The CIFAR 100 dataset consists of 60000 32x32 color images. These images are categorized into 100 classes. The 100 classes are further grouped into 20 superclasses. We have mentioned all the classes and superclasses in the Appendix - Exhibit 1. The training set contains 50000 images with 500 images per class. The test set includes 100 pictures for each category for a total of 10000 images. The training and test sets occupy 148 MB and 29.6 MB of storage space, respectively. The data is stored as a NumPy array of uint8s. Each row stores a 32x32 image. We have attached some sample images in Fig 1. From Fig1, we can see that the pictures are relatively small and blurry, which makes the classification task challenging as algorithms like CNN struggle with objects that are small or thin. Furthermore, the dataset also has some wrong labels. E.g., the first image on the second row is labeled as a lion, but it is a tiger.

(a) Fig 1: Sample Images from the Dataset



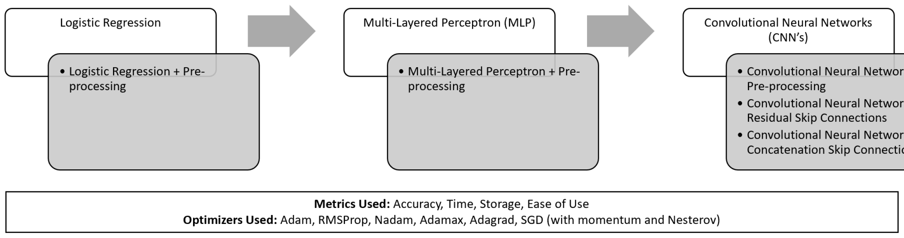
(b) Fig 2: Input Image is passed to model which predicts its class

1.4 Plan for Analysis and Questions to Answer

- How do the different optimizers like Adam, RMSProp, Nadam, Adamax, Adagrad, SGD (with momentum and nesterov) compare with each other on the different evaluation metrics?
- How do different machine learning techniques like logistic regression, SVM, MLP and CNN compare with each other on the evaluation metrics?
- What is the impact of image transformations like centering and standardizing (pre-processing the data) on the model performance?

2 Methodology

2.1 Methods



Logistic Regression
• Logistic Regression + Pre-processing
Multi-Layered Perceptron (MLP)
• Multi-Layered Perceptron + Pre-processing
Convolutional Neural Networks (CNN's)
• Convolutional Neural Networks + Pre-processing
• Convolutional Neural Networks + Residual Skip Connections
• Convolutional Neural Networks + Concatenation Skip Connections
Metrics Used: Accuracy, Time, Storage, Ease of Use
Optimizers Used: Adam, RMSProp, Nadam, Adamax, Adagrad, SGD (with momentum and Nesterov)

(a) Fig 3: High-Level Diagram of the Entire Flow of Experiment

Our workflow is described in Fig 3. We begin our analysis with logistic regression (LR) as our baseline model. Logistic regression is a linear classifier and is one of the simplest yet effective classification algorithms. Our second technique is Support Vector Machines (SVM). SVMs were widely used for image classifications before neural networks gained prominence. Then we moved to neural networks.

Neural networks have achieved human-level performance in many tasks, including image classification, with their ability to improve through experience automatically. So, we decided to use neural networks. We start with a multi-layered perceptron (MLP) as it has more dense layers compared to just one layer for the LR, and we expect it to perform better than the previous models. CNNs are widely used in image classification tasks and can outperform humans at classifying objects into fine-grained categories. We expect the CNN model to have good performance in the image classification task on the CIFAR dataset with 100 fine-grained classes. Furthermore, we implemented CNNs with concatenation and residual skip connections. Residual and concatenation skip connections form the backbone of the widely used ResNets and DenseNets, respectively.

Additionally, we repeated the model training of all the above methods by pre-processing the data. We measured all these techniques' performance on the evaluation metrics described below with different optimization techniques like Adam, RMSProp, Nadam, Adamax, Adagrad, and SGD (with momentum and Nesterov). We have attached the models' architecture block diagrams in the Appendix - Exhibit 2 to 6.

Our pipeline (Fig 2) involves passing an image through a machine learning algorithm. The algorithm outputs the most likely class for the image. We are using the SoftMax activation function to give us the probability distribution over the classes. The training pipeline involves loss computation, and the error is then back-propagated through the network to adjust the weights. We are calculating the loss with categorical cross entropy.

2.2 Metrics

We evaluated the models' performance on the following metrics: accuracy, time, and storage. Accuracy is the number of correct predictions made by the model overall predictions made. It is a useful metric, especially when the classes in the data are nearly balanced. We measure the training time per epoch for the models. It gives us an insight into the model complexity. Complex models would take longer to train. The number of parameters of a model provides us with the storage for the model. Models with many parameters would require higher storage space. These three metrics would enable us to understand the cost and performance trade-off between different machine learning techniques and answer all the questions in our plan.

3 Computation

3.1 Resources and Tools Used

We performed our experiments on Google Colaboratory. Colab provides free access to GPUs. Hence, we did not incur any cost. We used only one machine to run our experiments. We wrote our experiments in Python and used Tensorflow and Keras as our computational platform with GPU. For SVM, we used the Skikit Learn Package. Colab automatically assigns a different GPU each time we start a new session. Hence, our experiments were performed on different GPUs like Tesla K80 and P40.

3.2 Overall Computational Cost

Our images are stored as a NumPy array of uints8 with each row containing a 32x32 image sample. We used a batch size of 64 for all our experiments. Storage costs in terms of number of parameters are discussed further in detail in the results section. The linear regression model was computationally the fastest with 2 seconds per epoch. The MLP and the vanilla CNN took 10 and 40 seconds per iteration, respectively. The CNN models with skip connections were the slowest (both concatenation and residual skip connections took over 3 minutes per epoch).

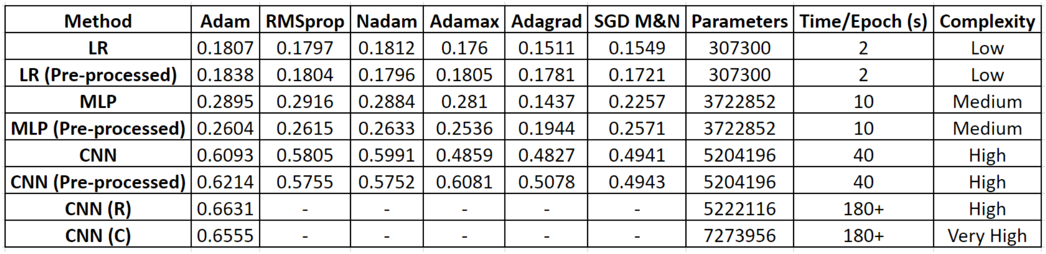
3.3 Computational Challenges

Our dataset is relatively small, occupying only 178 MB of storage space. Hence, we did not face any computational challenge in terms of time/storage while performing experiments. However, we did encounter frequent disconnections while using the free version of Colab. To overcome this challenge, we created multiple google accounts to run our experiments.

4 Results

The summary table in Fig 3 shows a performance comparison of different methods for various optimization techniques on the evaluation metrics. The numbers in the table for these optimization techniques represent the test accuracy. We can also visualize the best test accuracy of these models in Fig 4. We have trained the logistic regression, MLP, and the CNNs with skip connections (CNN (R) and CNN (C)) models for 50 iterations and the vanilla CNN for 100 iterations.

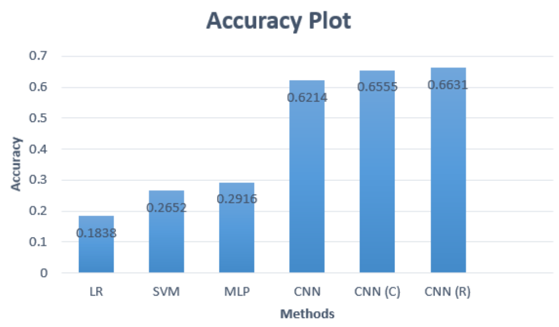
Figure 1: Summary Table of Different Methods for Optimization Techniques on Evaluation Metrics



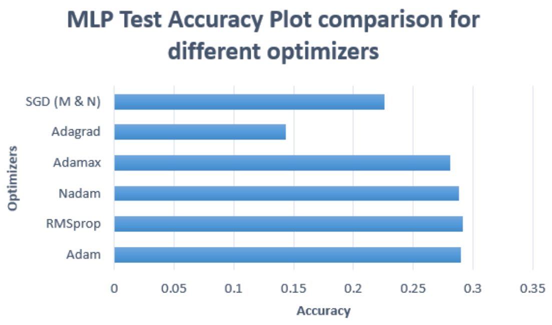
Method	Adam	RMSprop	Nadam	Adamax	Adagrad	SGD M&N	Parameters	Time/Epoch (s)	Complexity
LR	0.1807	0.1797	0.1812	0.176	0.1511	0.1549	307300	2	Low
LR (Pre-processed)	0.1838	0.1804	0.1796	0.1805	0.1781	0.1721	307300	2	Low
MLP	0.2895	0.2916	0.2884	0.281	0.1437	0.2257	3722852	10	Medium
MLP (Pre-processed)	0.2604	0.2615	0.2633	0.2536	0.1944	0.2571	3722852	10	Medium
CNN	0.6093	0.5805	0.5991	0.4859	0.4827	0.4941	5204196	40	High
CNN (Pre-processed)	0.6214	0.5755	0.5752	0.6081	0.5078	0.4943	5204196	40	High
CNN (R)	0.6631	-	-	-	-	-	5222116	180+	High
CNN (C)	0.6555	-	-	-	-	-	7273956	180+	Very High

Logistic regression (LR) has the lowest number of parameters and an accuracy of 0.1838. It has the most insufficient accuracy and the fastest training time per epoch amongst all our techniques. The SVM model has a modest test accuracy of 0.2652. SVM is significantly slower to train. The MLP improves the test accuracy to 0.2916 and has a much faster training time till convergence compared to SVM. Its total parameters are an order of magnitude higher than logistic regression.

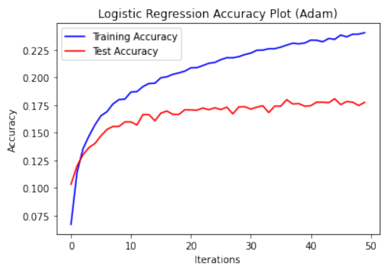
Vanilla CNN substantially improves test accuracy to 0.6214 with significantly more parameters than MLP. Both the skip connection CNNs have better test accuracy than vanilla CNN. CNN with residual



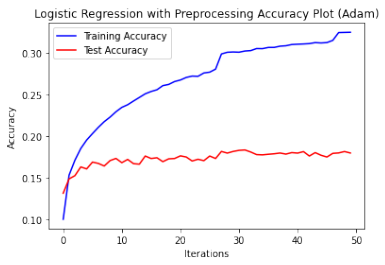
(a) Fig 4: Best Test Accuracy Plot



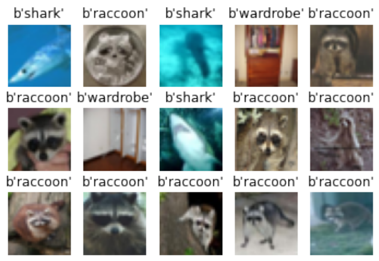
(b) Fig 5: Optimizers Comparison Plot for MLP



(c) Fig 6: LR Accuracy Plot



(d) Fig 7: LR Accuracy plot (Pre-processed)



(e) Fig 8: Error Analysis

connections (CNN (R)) improves test accuracy by more than 4 percent points (0.6631) than vanilla CNN with almost the same number of parameters. CNN with concatenation skip connections (CNN (C)) also has a higher test accuracy than CNN but has much more parameters. Both CNN (R) and CNN (C) are complex models and take more time per iteration. Although both models converge within 50 iterations compared to CNN's 100, the total time to train the CNN model for 100 iterations is less than 70 minutes, while the models with skip connections take more than 150 minutes to train for 50 epochs.

Hence, if the model's accuracy is more important, we will choose CNN with residual skip connections. But if training and inference times are more critical, then we will select the vanilla CNN model.

4.1 Analysis of Different Optimizers

From Fig 5, we can see that Adam, Nadam and RMSprop have comparable performances for the MLP model. Adamax and SGD with momentum and Nesterov (SGD(M&N)) have a slightly lower performance. Adagrad has the lowest performance. All models were trained for the same number of epochs at optimized learning rates. Similar trend is observed across other methods as well.

4.2 Analysis of Pre-processing data

From Fig 7, we can see that image transformations like centering and standardizing leads to faster convergence for the linear regression model. The test accuracy after 50 epochs for both the models (Fig 6 and 7) are similar. But the model trained without transformation starts from a lower test accuracy and takes longer to converge compared to the model with image transformation. Similar trend is observed across other methods. Hence, pre-processing is beneficial for faster convergence.

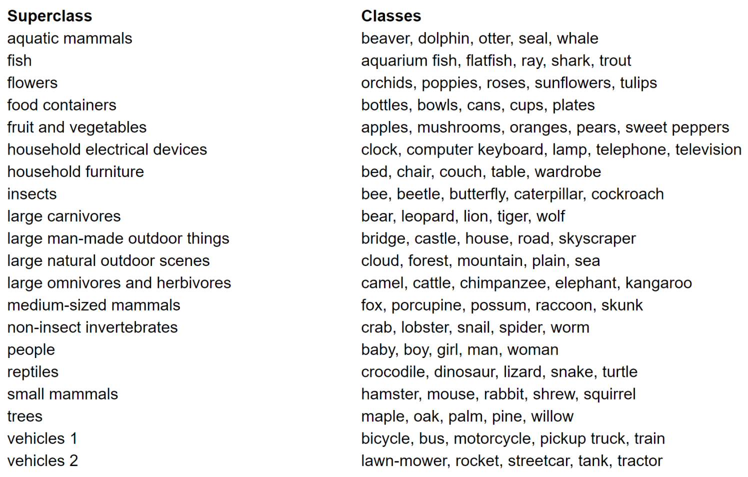
4.3 Error Analysis

Our best model made maximum errors while predicting racoon, shark, and wardrobe (Fig 8). The model could only correctly predict 32 racoons, 30 sharks and 38 wardrobes out of 100. It is difficult to ascertain the reasons for such low performance on these classes as the wrong predictions were scattered across different classes.

5 Appendix

5.1 Exhibit 1 - Classes and Superclasses in the CIFAR 100 Dataset

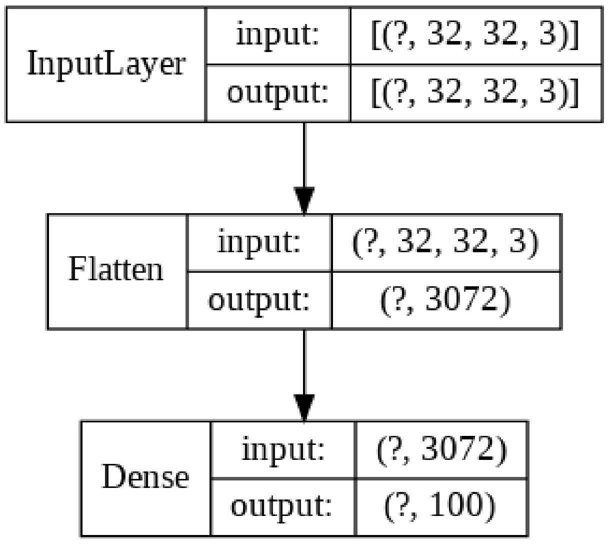
Figure 2: Classes and Superclasses in the CIFAR 100 Dataset



Superclass	**Classes**
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

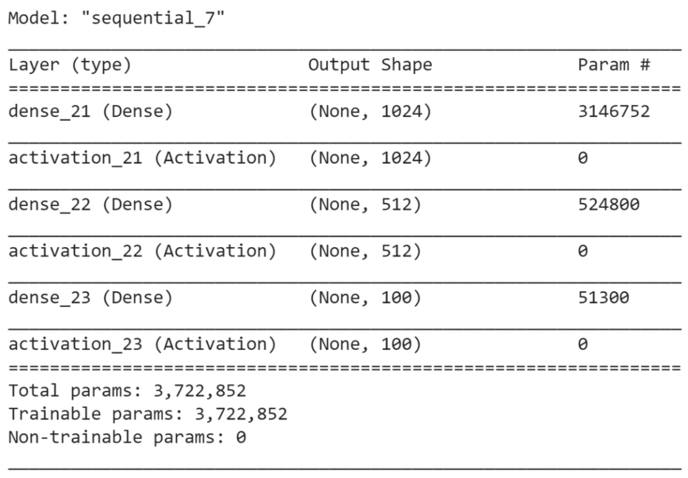
5.2 Exhibit 2 - Logistic Regression Model Architecture

Figure 3: Logistic Regression Model Architecture



5.3 Exhibit 3 - Multi-Layered Perceptron Model Architecture

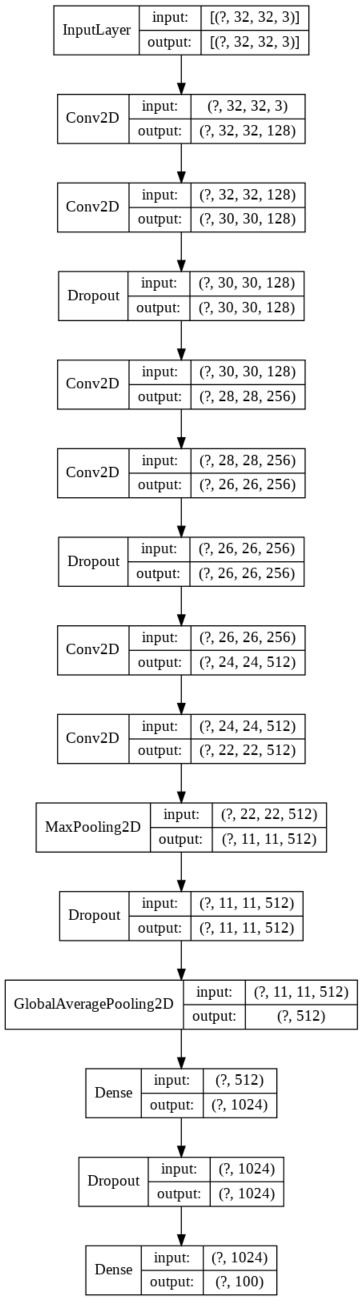
Figure 4: Multi-Layered Perceptron Model Architecture



```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_21 (Dense)             (None, 1024)              3146752
_________________________________________________________________
activation_21 (Activation)   (None, 1024)              0
_________________________________________________________________
dense_22 (Dense)             (None, 512)               524800
_________________________________________________________________
activation_22 (Activation)   (None, 512)               0
_________________________________________________________________
dense_23 (Dense)             (None, 100)               51300
_________________________________________________________________
activation_23 (Activation)   (None, 100)               0
=================================================================
Total params: 3,722,852
Trainable params: 3,722,852
Non-trainable params: 0
_________________________________________________________________
```

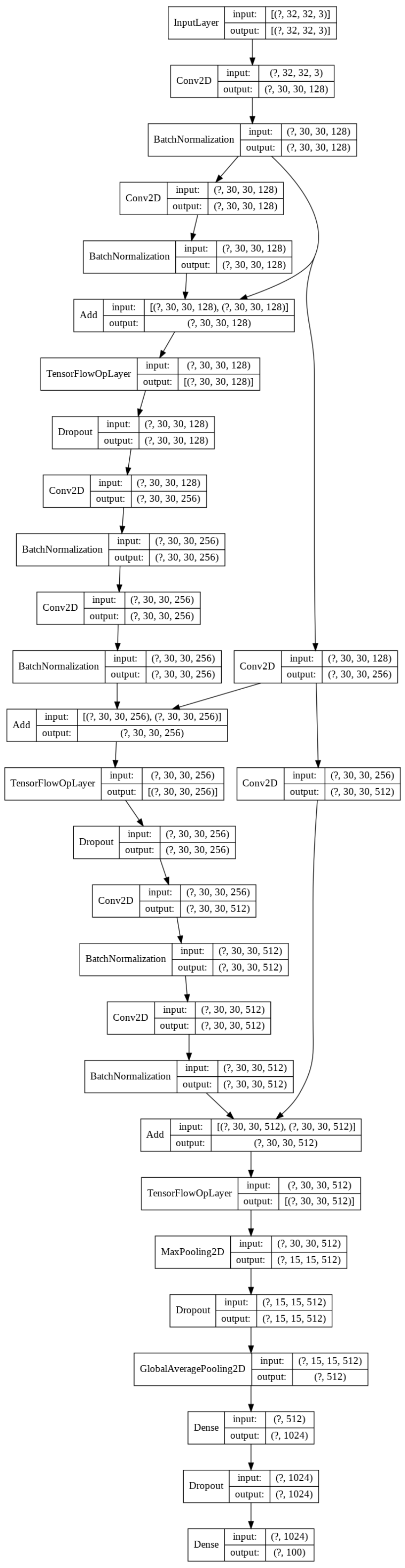
5.4 Exhibit 4 - Convolutional Neural Network Model Architecture

Figure 5: Convolutional Neural Network Model Architecture


InputLayer | input: [(?, 32, 32, 3)] | output: [(?, 32, 32, 3)]
Conv2D | input: (?, 32, 32, 3) | output: (?, 32, 32, 128)
Conv2D | input: (?, 32, 32, 128) | output: (?, 30, 30, 128)
Dropout | input: (?, 30, 30, 128) | output: (?, 30, 30, 128)
Conv2D | input: (?, 30, 30, 128) | output: (?, 28, 28, 256)
Conv2D | input: (?, 28, 28, 256) | output: (?, 26, 26, 256)
Dropout | input: (?, 26, 26, 256) | output: (?, 26, 26, 256)
Conv2D | input: (?, 26, 26, 256) | output: (?, 24, 24, 512)
Conv2D | input: (?, 24, 24, 512) | output: (?, 22, 22, 512)
MaxPooling2D | input: (?, 22, 22, 512) | output: (?, 11, 11, 512)
Dropout | input: (?, 11, 11, 512) | output: (?, 11, 11, 512)
GlobalAveragePooling2D | input: (?, 11, 11, 512) | output: (?, 512)
Dense | input: (?, 512) | output: (?, 1024)
Dropout | input: (?, 1024) | output: (?, 1024)
Dense | input: (?, 1024) | output: (?, 100)

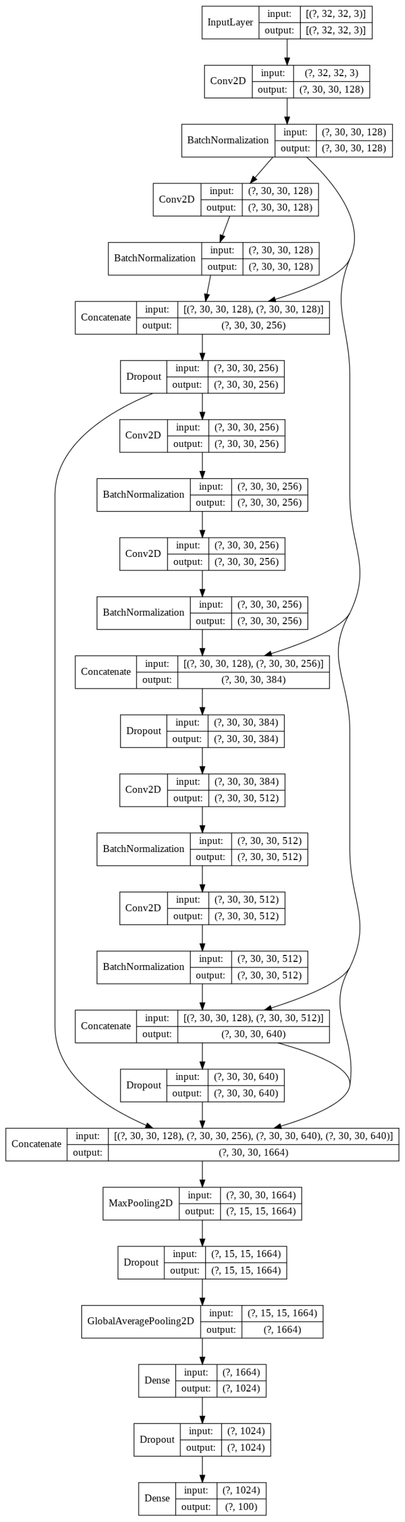
5.5 Exhibit 5 - Convolutional Neural Network Residual Skip Connection Architecture

Figure 6: CNN (R) Model Architecture


InputLayer input: [(?, 32, 32, 3)] output: [(?, 32, 32, 3)]
Conv2D input: (?, 32, 32, 3) output: (?, 30, 30, 128)
BatchNormalization input: (?, 30, 30, 128) output: (?, 30, 30, 128)
Conv2D input: (?, 30, 30, 128) output: (?, 30, 30, 128)
BatchNormalization input: (?, 30, 30, 128) output: (?, 30, 30, 128)
Add input: [(?, 30, 30, 128), (?, 30, 30, 128)] output: (?, 30, 30, 128)
TensorFlowOpLayer input: (?, 30, 30, 128) output: [(?, 30, 30, 128)]
Dropout input: (?, 30, 30, 128) output: (?, 30, 30, 128)
Conv2D input: (?, 30, 30, 128) output: (?, 30, 30, 256)
BatchNormalization input: (?, 30, 30, 256) output: (?, 30, 30, 256)
Conv2D input: (?, 30, 30, 256) output: (?, 30, 30, 256)
BatchNormalization input: (?, 30, 30, 256) output: (?, 30, 30, 256)
Conv2D input: (?, 30, 30, 128) output: (?, 30, 30, 256)
Add input: [(?, 30, 30, 256), (?, 30, 30, 256)] output: (?, 30, 30, 256)
TensorFlowOpLayer input: (?, 30, 30, 256) output: [(?, 30, 30, 256)]
Conv2D input: (?, 30, 30, 256) output: (?, 30, 30, 512)
Dropout input: (?, 30, 30, 256) output: (?, 30, 30, 256)
Conv2D input: (?, 30, 30, 256) output: (?, 30, 30, 512)
BatchNormalization input: (?, 30, 30, 512) output: (?, 30, 30, 512)
Conv2D input: (?, 30, 30, 512) output: (?, 30, 30, 512)
BatchNormalization input: (?, 30, 30, 512) output: (?, 30, 30, 512)
Add input: [(?, 30, 30, 512), (?, 30, 30, 512)] output: (?, 30, 30, 512)
TensorFlowOpLayer input: (?, 30, 30, 512) output: [(?, 30, 30, 512)]
MaxPooling2D input: (?, 30, 30, 512) output: (?, 15, 15, 512)
Dropout input: (?, 15, 15, 512) output: (?, 15, 15, 512)
GlobalAveragePooling2D input: (?, 15, 15, 512) output: (?, 512)
Dense input: (?, 512) output: (?, 1024)
Dropout input: (?, 1024) output: (?, 1024)
Dense input: (?, 1024) output: (?, 100)

5.6 Exhibit 6 - Convolutional Neural Network Concatenated Skip Connection Architecture

Figure 7: CNN (C) Model Architecture


InputLayer input: [(?, 32, 32, 3)] output: [(?, 32, 32, 3)]
Conv2D input: (?, 32, 32, 3) output: (?, 30, 30, 128)
BatchNormalization input: (?, 30, 30, 128) output: (?, 30, 30, 128)
Conv2D input: (?, 30, 30, 128) output: (?, 30, 30, 128)
BatchNormalization input: (?, 30, 30, 128) output: (?, 30, 30, 128)
Concatenate input: [(?, 30, 30, 128), (?, 30, 30, 128)] output: (?, 30, 30, 256)
Dropout input: (?, 30, 30, 256) output: (?, 30, 30, 256)
Conv2D input: (?, 30, 30, 256) output: (?, 30, 30, 256)
BatchNormalization input: (?, 30, 30, 256) output: (?, 30, 30, 256)
Conv2D input: (?, 30, 30, 256) output: (?, 30, 30, 256)
BatchNormalization input: (?, 30, 30, 256) output: (?, 30, 30, 256)
Concatenate input: [(?, 30, 30, 128), (?, 30, 30, 256)] output: (?, 30, 30, 384)
Dropout input: (?, 30, 30, 384) output: (?, 30, 30, 384)
Conv2D input: (?, 30, 30, 384) output: (?, 30, 30, 512)
BatchNormalization input: (?, 30, 30, 512) output: (?, 30, 30, 512)
Conv2D input: (?, 30, 30, 512) output: (?, 30, 30, 512)
BatchNormalization input: (?, 30, 30, 512) output: (?, 30, 30, 512)
Concatenate input: [(?, 30, 30, 128), (?, 30, 30, 512)] output: (?, 30, 30, 640)
Dropout input: (?, 30, 30, 640) output: (?, 30, 30, 640)
Concatenate input: [(?, 30, 30, 128), (?, 30, 30, 256), (?, 30, 30, 640), (?, 30, 30, 640)] output: (?, 30, 30, 1664)
MaxPooling2D input: (?, 30, 30, 1664) output: (?, 15, 15, 1664)
Dropout input: (?, 15, 15, 1664) output: (?, 15, 15, 1664)
GlobalAveragePooling2D input: (?, 15, 15, 1664) output: (?, 1664)
Dense input: (?, 1664) output: (?, 1024)
Dropout input: (?, 1024) output: (?, 1024)
Dense input: (?, 1024) output: (?, 100)