Computer Vision: CIFAR-10 Data

Dhaval Patel

Northwestern University

Abstract

The objective of this research is to analyze the features that make convolutional neural networks effective in image classification, particularly in medical imaging diagnostics. We evaluated several models and compare deep neural network models to convolutional neural networks. We conducted 13 experiments on different models to examine how they performed in image classification. To classify the images in the Canadian Institute for Advanced Research (CIFAR-10) database we employed a multi-classification problem with 10 classes as the output and a deep learning model trained with the use of convolutional neural networks.

Introduction

In 1959, two neurophysiologists, Hubel and Weisel, found that when they presented a cat with a range of images, the cat's brain responded first by detecting the hard edges or lines (Rostyslav, 2019). This prompted the Japanese computer scientist Kunihiko Fukushima to create the "neocognitron" in 1980, which is a hierarchical, multilayered artificial neural network. This neural network comprised of convolutional and down sampling layers that could perform fundamental image processing tasks. Using a CNN model inspired by the neocognitron, Ciresan et al. for the first time it achieved human-like performance in 2012 on the MNIST data set and outperformed humans by a factor of two on the traffic signs dataset.

Deep learning has grown in prominence during the last decade. Machine-learning techniques, particularly convolutional neural networks are being used in the medical industry to improve delivery of patient care services. CNN has been widely used in medical imaging classification, object identification, and semantic segmentation. Machine learning techniques in detecting COVID-19 and the potential of Neural Networks and DL for the task developed an on-device COVID-19 screening system that uses X-ray images to identify COVID-19 infections as illustrated below in figure 1. The CNN architecture was able to learn image-level features while the GCN would learn the relation-aware representation (RAR) features (Kumar et al., 2021).

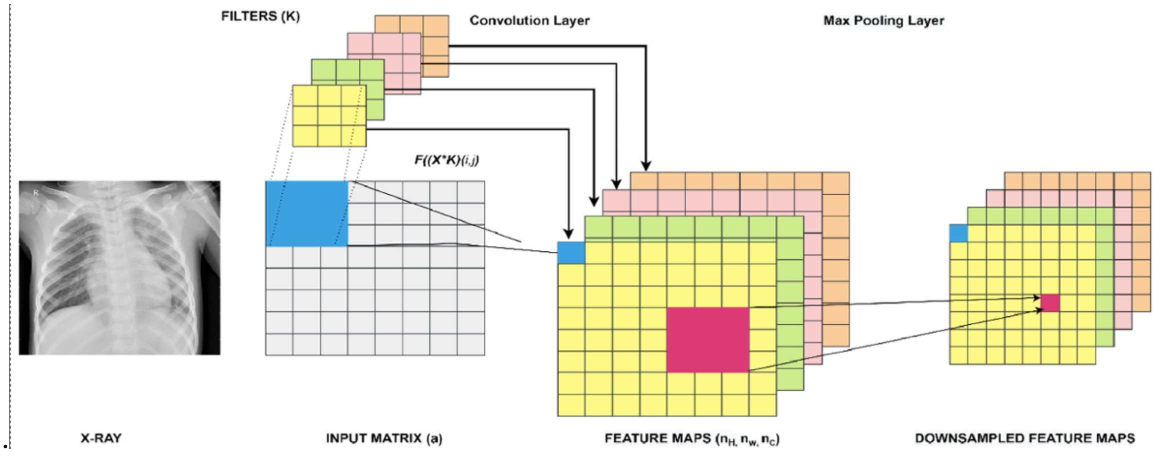
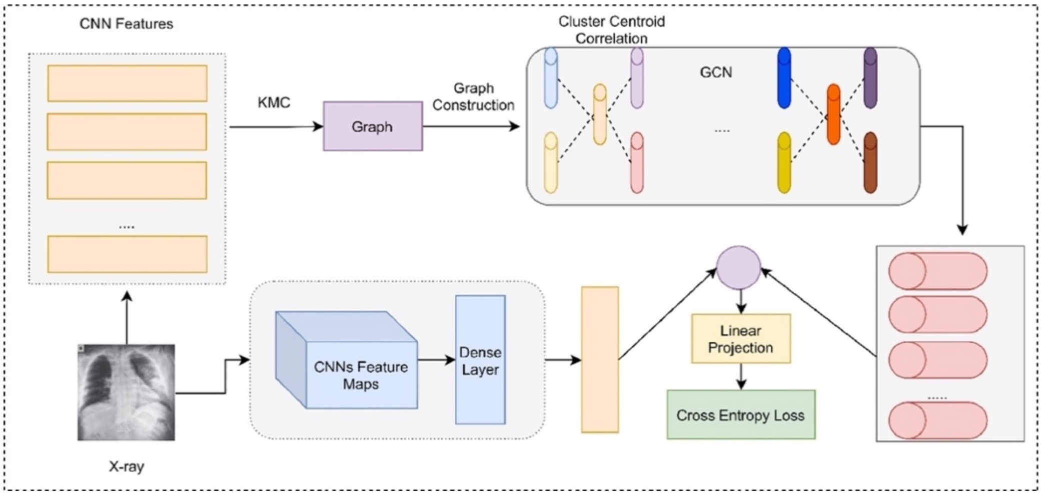


Fig.1

Literature Review

Since 2020, COVID-19 has become one of the worst pandemics. Screening techniques like RT-PCR are the best way to detect severe acute respiratory syndrome coronavirus in patients since at-home quick tests are inaccurate. These tests require lab testing, which takes time. Early research has revealed that visual indications in patients' chest X-rays or Computer tomography (CT) images can be used to identify Covid-19 patients and track the virus's transmission across the community. Inspired by this, the researchers created SARS-Net, a deep learning network that can categorize and detect Covid-19-diagnosed chest X-rays (Kumar et al., 2021).

The researchers demonstrated in their study that the novel architecture outperformed current framework approaches in terms of accuracy. On the validation set, the suggested model has an accuracy of 97.60% and a sensitivity of 92.90%. The SARS-Net design begins with a CNN that employs parallel concatenation in a single block known as the Inception block, as well as certain convolution layers. It is enhanced by Coordinate-Convolutions (CC), anti-aliased (AA) CNNs, and rank-based stochastic pooling (RSP). The final SAR-Net model is created by combining it with a 2-layer graph convolution network as represented below (Kumar et al., 2021)



(Fig.2)

One reason that because SARS-Net is partially made of a CNN, which is our core architecture, therefore this study would provide significant insights into the best hyperparameters to modify our model. Furthermore, because SARS-net is likely to outperform conventional CNNs, this architecture might be a viable choice to evaluate our dataset as part of future recommendations. This is a Computer-aided Diagnostic (CADx) system, which uses a mix of graph convolutional networks (GCN) and convolutional neural networks (CNN) to detect irregularities in Chest X-Ray images that might indicate the existence of Covid-19 infections in patients (Kumar, et.al 2021). The advancements in deep learning algorithms and the open-source datasets accessible for experimentation are responsible for the CNN's performance in image recognition. To obtain a better understanding of CNN, this study will experiment with several methods of training such a network on the CIFAR-10 dataset.

Methods

Review Research

This research study explored a multi-classification problem with 10 output classes, where the deep learning model learns to detect and classify images using CNN and DNN. CNN is comparable to conventional neural networks in that it is composed of neurons with learnable weights, but it varies from MLP as it relies on the weight sharing principle. There are three types of CNN networks described in the literature: one-dimensional CNN (Conv1D), two-dimensional CNN (Conv2D), and three-dimensional CNN (Conv3D).

Convolutional Neural Network Model Creation Process

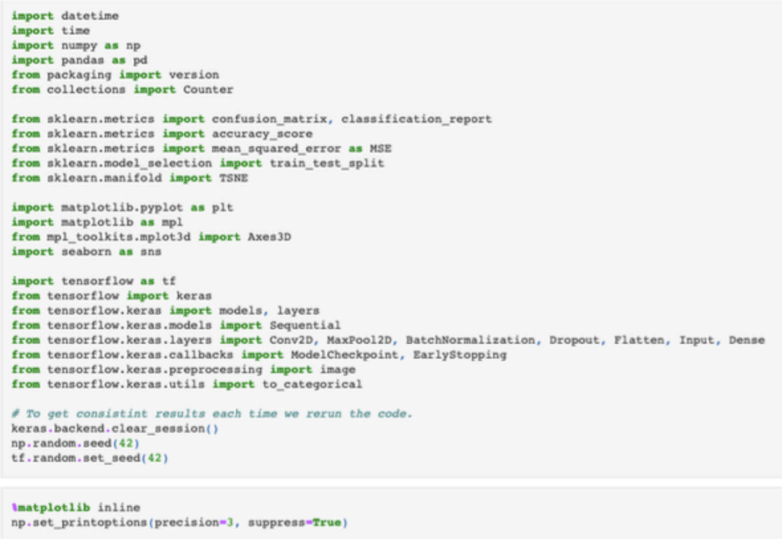
We utilized Conv2D, which is often used for image data with 3-dimensional input and output. CNNs have three sequential layers as follows: convolution, pooling, and non-linearity followed by several fully connected layers. Convolutional operation drives CNN as it reduces the image while maintaining its features and removing noise. Sliding a convolution filter across input data creates a feature map. The filters are the only parameters automatically learnt throughout the training process, whereas the size and quantity may be specified up. Filters are 3x3, 5x5, or 7x7. Small filters can extract more intensely local information from input. As we move further into the network, we may increase the number of filters for a more powerful model, but we risk overfitting due to higher parameter count.

Stride determines how far the filter moves across the input at each step. As we move the filter across the input, we do element-wise matrix multiplication and add the results, which are then stored in the feature map. Padding can also be used to conserve the size of feature maps by surrounding the input with zero values.

On an input, multiple convolutions layers are performed, each employing a different filter and producing distinct feature maps. Following the convolution procedure, we run the output via an activation function and pooling (typically, max pooling, which takes the maximum value in the pooling window) to lower the dimensionality of each feature map individually, reducing the height and width while maintaining the depth. This stage decreases the number of parameters, reducing training time and combating overfitting.

Convolution layers learn such complicated characteristics by stacking them on top of one another. The initial layers identify edges or corners, the subsequent levels combine these to recognize forms, and the subsequent layers integrate this information to infer that this is a face of object. We then flatten the output and add a couple of fully connected layers in a procedure like that of a feed-forward network.

As shown in figure 3 below, we imported all the packages for data visualization, creation, and testing for both DNN and CNN models.



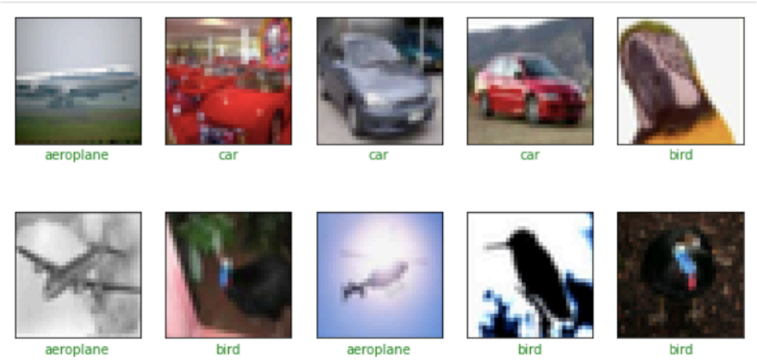
```
import datetime
import time
import numpy as np
import pandas as pd
from packaging import version
from collections import Counter

from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error as MSE
from sklearn.model_selection import train_test_split
from sklearn.manifold import TSNE

import matplotlib.pyplot as plt
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import models, layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPool2D, BatchNormalization, Dropout, Flatten, Input, Dense
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.preprocessing import image
from tensorflow.keras.utils import to_categorical

# To get consistint results each time we rerun the code.
keras.backend.clear_session()
np.random.seed(42)
tf.random.set_seed(42)
```

```
%matplotlib inline
np.set_printoptions(precision=3, suppress=True)
```

(Fig.3 Packages)

Data Preparation, Exploration, Visualization Process

Using TensorFlow Keras, the CIFAR-10 dataset was loaded. The dataset contains 60,000 images that have previously been classified, including 50,000 labeled images for the training dataset and 10,000 labeled images for the testing dataset. For validation, 5000 images were selected from the training dataset. There is no difference in label distribution between the three datasets. Airplanes, vehicles, birds, cats, deer, dogs, frogs, horses, ships, and trucks are represented by the ten different classes. All images are color and 32x32x3 pixels in size. Each pixel has a number between 0 and 255 that describes its intensity, with 255 being maximum brightness and 0 representing turned off. Figure 4 depicts a random selection of 10 images and their labels.



(Fig.4 Sample Images)

Data Regularization and Compilation Process

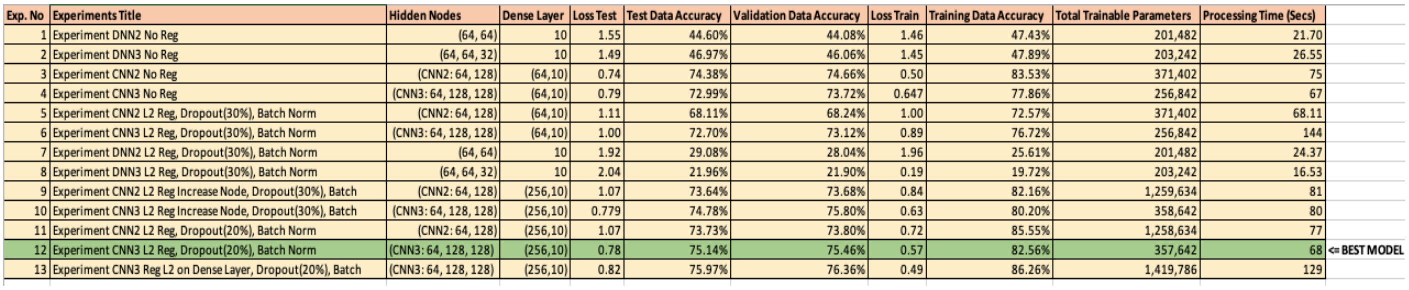
The methods used for data regularization are as follows: L2 regularize, dropout rate, early stopping, and batch normalization are crucial from preventing the model overfitting, model complexity, and avoid vanishing or exploding gradients. In L2 regularization to minimize the loss function, the algorithm should minimize both original the loss function plus the regularization term, which depends on the square of the weights. As we increase lambda value, the parameters value will decrease as L2 will penalize the parameters. The difference between L1 and L2 regularize is that the weights will not be sparse, and we will get much better accuracy values compared to L1. Similarly, the dropout rate randomly selects some nodes and removes them every iteration with all their incoming and outgoing connections. Finally, batch normalization technique helps improving the performance and stability of neural networks by normalizing the inputs of each layer in such a way that they have a mean output of zero and standard deviation of one. So, we get normalized output of one layer before applying the activation function, and then feed it into the other layers. To prepare the network for training, we needed to establish the algorithms that will use to optimize the weights and biases based on the data. We compiled the model with:

- The sparse categorical crossentropy is commonly used and loss function provides the difference between the predicted outputs and the actual outputs given in the dataset. The goal is to minimize the difference between these two distributions.
- The activation function used is Adam, which is a combination of RMSprop and Stochastic Gradient Descent with momentum. It an optimizer that allows the network to update itself depending on the data it sees and its loss function by keeping a moving average of the squared gradient for each weight.

- The softmax activation function of the last layer adds up the evidence that an input is in a specific class by doing a weighted sum of the pixel intensity. The weight is negative if that pixel having a high intensity is evidence against the image being in that class, and positive if it is evidence in favor. Then, it converts that evidence into 10 probability scores, summing up to 1.

Results

DNN and CNN Model Experiments Results (Fig.5)



Exp. No	Experiments Title	Hidden Nodes	Dense Layer	Loss Test	Test Data Accuracy	Validation Data Accuracy	Loss Train	Training Data Accuracy	Total Trainable Parameters	Processing Time (Secs)	
1	Experiment DNN2 No Reg	(64, 64)	10	1.55	44.60%	44.08%	1.46	47.43%	201,482	21.70	
2	Experiment DNN3 No Reg	(64, 64, 32)	10	1.49	46.97%	46.06%	1.45	47.89%	203,242	26.55	
3	Experiment CNN2 No Reg	(CNN2: 64, 128)	(64,10)	0.74	74.38%	74.66%	0.50	83.53%	371,402	75	
4	Experiment CNN3 No Reg	(CNN3: 64, 128, 128)	(64,10)	0.79	72.99%	73.72%	0.647	77.86%	256,842	67	
5	Experiment CNN2 L2 Reg, Dropout(30%), Batch Norm	(CNN2: 64, 128)	(64,10)	1.11	68.11%	68.24%	1.00	72.57%	371,402	68.11	
6	Experiment CNN3 L2 Reg, Dropout(30%), Batch Norm	(CNN3: 64, 128, 128)	(64,10)	1.00	72.70%	73.12%	0.89	76.72%	256,842	144	
7	Experiment DNN2 L2 Reg, Dropout(30%), Batch Norm	(64, 64)	10	1.92	29.08%	28.04%	1.96	25.61%	201,482	24.37	
8	Experiment DNN3 L2 Reg, Dropout(30%), Batch Norm	(64, 64, 32)	10	2.04	21.96%	21.90%	0.19	19.72%	203,242	16.53	
9	Experiment CNN2 L2 Reg Increase Node, Dropout(30%), Batch	(CNN2: 64, 128)	(256,10)	1.07	73.64%	73.68%	0.84	82.16%	1,259,634	81	
10	Experiment CNN3 L2 Reg Increase Node, Dropout(30%), Batch	(CNN3: 64, 128, 128)	(256,10)	0.779	74.78%	75.80%	0.63	80.20%	358,642	80	
11	Experiment CNN2 L2 Reg, Dropout(20%), Batch Norm	(CNN2: 64, 128)	(256,10)	1.07	73.73%	73.80%	0.72	85.55%	1,258,634	77	
12	Experiment CNN3 L2 Reg, Dropout(20%), Batch Norm	(CNN3: 64, 128, 128)	(256,10)	0.78	75.14%	75.46%	0.57	82.56%	357,642	68	<= BEST MODEL
13	Experiment CNN3 Reg L2 on Dense Layer, Dropout(20%), Batch	(CNN3: 64, 128, 128)	(256,10)	0.82	75.97%	76.36%	0.49	86.26%	1,419,786	129	

We conducted 13 experiments as shown in the figure 5 above. The first 4 experiments we did not use regularization for DNN and CNN2D models with max pooling layers that had variations 2 and 3 layers running with batch size of 100 for each iteration and 20 epochs for all experiments. For experiments 5 to 8, we use regularization such as L2 Regularizes, Dropout, and Early Stopping, and Batch Normalization. Furthermore, experimentation from 9 to 11 involved tweaking hypermeters such as L2 Regularizes, Dropout rate, and increasing hidden to find best model with optimal accuracy, faster processing time, and appropriate fitting of the data. We will analysis the results of each experiment at a high level to understand how the accuracy for the DNN and CNN models have improved over time based on variation in experiments.

In **Experiment 1 and 2** we built DNN model with variations of 2 and 3 dense layers with 64 activated hidden neuron nodes in the activation function, where with the following columns are utilized for training instead of 3072 input feature dimensional space with no regularization, and we got accuracy of below 50% for both experiments on test dataset it is underfitting in predicting those 10 classes. In addition, there were only trainable around 204,000 parameters more error amount in the surface, it should be simpler and faster average processing time of 23 seconds to train the model in predicting the classes. In **Experiment 7 and 8**, we built DNN models with L2 regularizes minimizes the loss function and penalize having around 204,000 trainable parameters which are less as we increase lambda values and 30% dropout which removing nodes every iteration to prevent overfitting the model led to worst DNN model accuracy of under 30% accuracy of test dataset among all experimentation.

In **Experiment 3 and 4**, we built 2D CNN model which is often used for image data with 3-dimensional input and output like the CIFAR-10 dataset. In experiment 3 and 4 we have variation of 2 and 3 Convolution and Max pooling layers with no regularization which indicates higher chances of overfitting the model. Whereas the strides are equal 2 which determines how far the filter moves across the input at each step doing element-wise matrix multiplication storing into feature map. The 2x2 kernel filters are the only parameters automatically learnt throughout the training process to extract more intensely local information from input. In **Experiment 3,** CNN model with 2 convolutional layers of 64, 128 hidden nodes, and dense layers of 64 nodes for image classification had 74.38% test accuracy with loss test error is 0.74 is which decent. In contrast, train loss train error is 0.50 which shows that model is overfitting with high variance, more complexity as well 300,000 with trainable parameters with built time of 75 seconds.

The **Experiment 4** 2D CNN model with 3 convolution layers 64, 128 and 128 nodes that has dense layers of 64 nodes for image classification resulted in test data accuracy of 73% which has similar observation to experiment 3 has we increased the number of convolution layers with no regularization complexity of the model decreasing, less processing time, and it less overfitting indicating but still high variance compared to experiment 3 while analyzing the loss error for train and test data.

Furthermore, we analyze the **Experiment 3** by extracting the outputs from 2 filters of the 2 max pooling layers and visualize them in a grid as images as shown in figure 6 below.

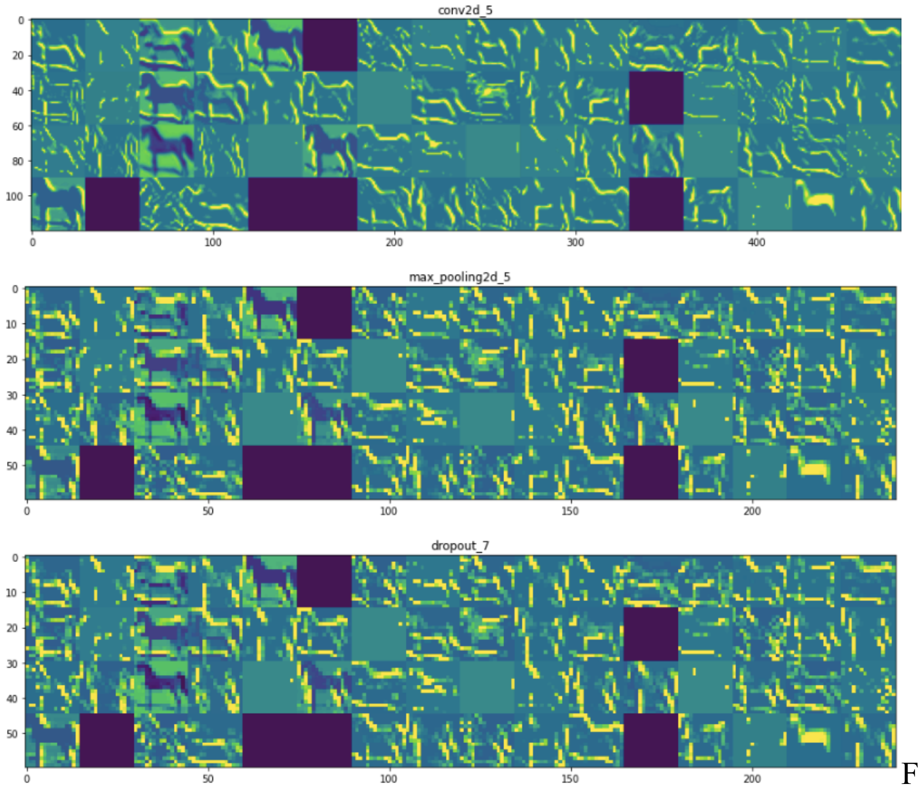


Fig.6

We can analyze the 'lighted' up regions as shown above correspond to some features in the original 'horse' images starting the convolution layer output followed by max pooling 2d and dropout layers tries to extract stripe like features which were diagonal lines to intensely gather more local information from input. The first layer seems to be responsible for detecting the edges of the horse and background. The learning becomes more abstract as we go down the layers and they become responsible for more specific detectors.

In **Experiment 5,** we conducted CNN model with 2 convolution and max pooling layers but with L2 regularization, 30% dropout rate, and batch normalization which would be used to prevent overfitting, stability in the neural network, and avoid vanishing or exploding gradients. The 2D CNN model with 2 convolutional layers has 64, 128 hidden nodes, and dense layers with 64 nodes for image classification had test data accuracy of 68.11% with loss test error is 1.11 is still high. In contrast, train loss train error is 1.00 which shows that model is almost appropriate fitting with high biases, model complexity and built time with trainable parameters is around as experiment 3 and 4 with no regularization.

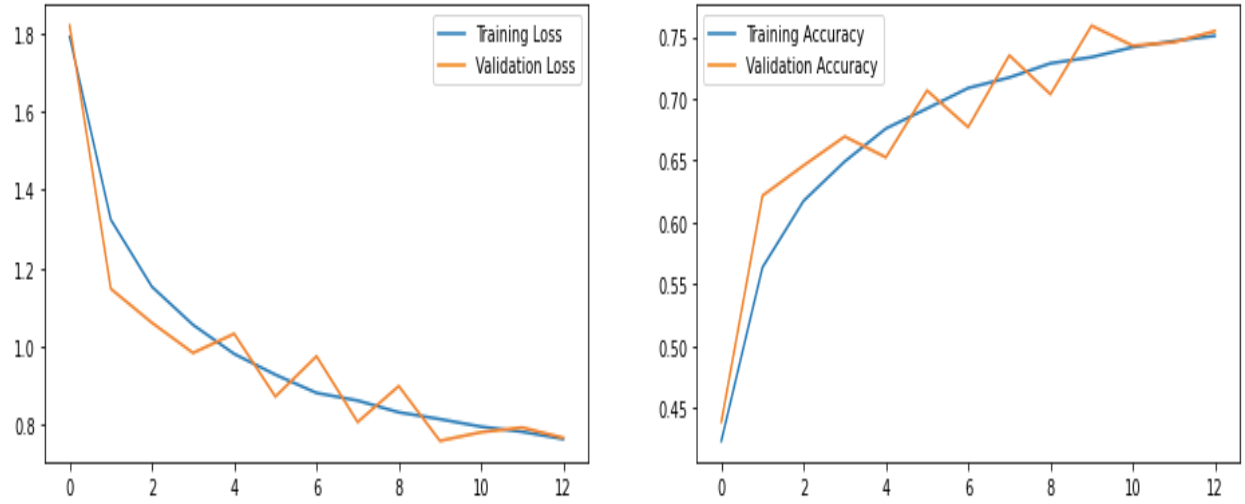
The **Experiment 6** consist of 2D CNN model with 3 convolution layers 64, 128 and 128 nodes that has dense layers of 64 nodes for image classification resulted in test data of 72.11% which has slightly higher by 3% compare to other experiments has we increase the number of convolution layers with regularization less complexity in the model and it less overfitting indicating but still low variance and biases compared to experiment 5 while analyzing the loss error for train and test data.

We conducted further experimentation from 9 to 13 to see if we can get better optimal accuracy with 2D CNN model following regularization to properly predicts all the 10 images classes. By tweaking hypermeters such as L2 Regularizes, 20% Dropout rate, and doing variations of 2 and 3 convolutional layers 64, 128 and 128 nodes that has dense layers of 256 nodes which did increase test accuracy from 73-74% for **experiments 9 to 11** but it increases model complexity with over 1.2 million trainable parameters and processing time to average 80 seconds to train models. We observed the low training error and high testing loss error indicating overfitting the models even after regularization because of increasing the hidden nodes.

Finally, after doing several experiments we found an optimal best final model that we can recommend to the business is **Experiment 12** based on the results in figure 5. Below are the training and validation plot showing higher accuracy and appropriate fitting of the data.

Best Final Model: Experiment 12 CNN3: (64,128, 128)

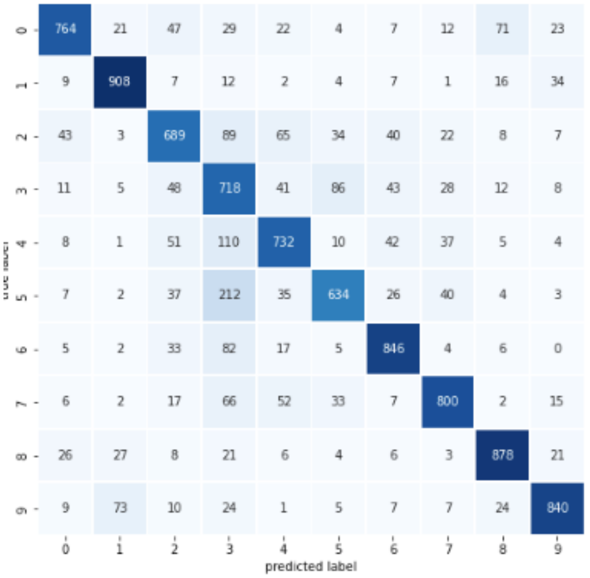
and Dense Layer (256, 10) Plot (Fig. 7)



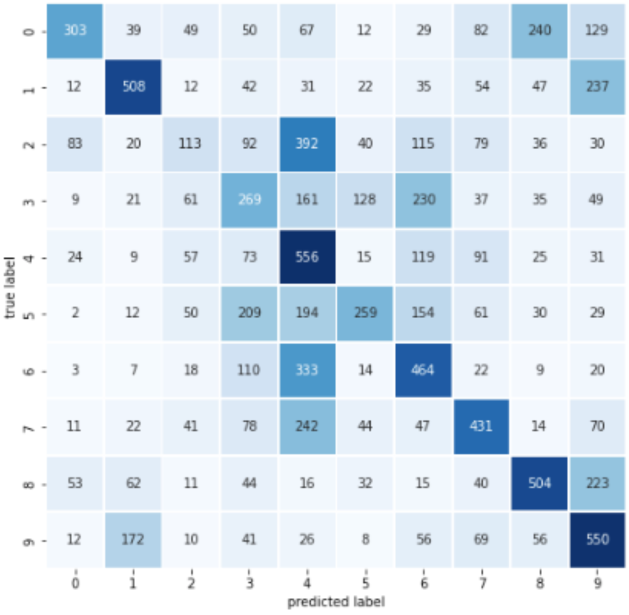
The **Experiment 12**, CNN model building process contains with 3 convolution layers 64, 128 and 128 nodes that has dense layers of 256 nodes following L2 regularize on dense layer only, 20 % dropout, batch normalization, and early stopping regularization techniques were used in the process of prevent model overfitting. The image classification resulted in test data of 75.14% which is only slightly higher but much better optimization wise compared to other experiments, but as we increasde the number of convolution layers with regularization which helped with complexity in the model and appropriate fitting indicating but still low variance and biases while analyzing the loss errors of dataset as well as based on the plot metrics from figure 7 shown above. The following experiment led to faster training time of 68 seconds because of only trainable 357,642 trainable parameters compared to other models. The complexity of model is low and has appropriate fitting. The other experiments 11 and 13 were for investigating variations towards improving accuracy, complexity, and performance of the CNN model.

Evaluate Model

Confusion Matrix (Best and Worst Model from Experimentation Results)



	0	1	2	3	4	5	6	7	8	9
0	764	21	47	29	22	4	7	12	71	23
1	9	908	7	12	2	4	7	1	16	34
2	43	3	689	89	65	34	40	22	8	7
3	11	5	48	718	41	86	43	28	12	8
4	8	1	51	110	732	10	42	37	5	4
5	7	2	37	212	35	634	26	40	4	3
6	5	2	33	82	17	5	846	4	6	0
7	6	2	17	66	52	33	7	800	2	15
8	26	27	8	21	6	4	6	3	878	21
9	9	73	10	24	1	5	7	7	24	840



	0	1	2	3	4	5	6	7	8	9
0	303	39	49	50	67	12	29	82	240	129
1	12	508	12	42	31	22	35	54	47	237
2	83	20	113	92	392	40	115	79	36	30
3	9	21	61	269	161	128	230	37	35	49
4	24	9	57	73	556	15	119	91	25	31
5	2	12	50	209	194	259	154	61	30	29
6	3	7	18	110	333	14	464	22	9	20
7	11	22	41	78	242	44	47	431	14	70
8	53	62	11	44	16	32	15	40	504	223
9	12	172	10	41	26	8	56	69	56	550

Fig.8 Best CNN Model Confusion Matrix

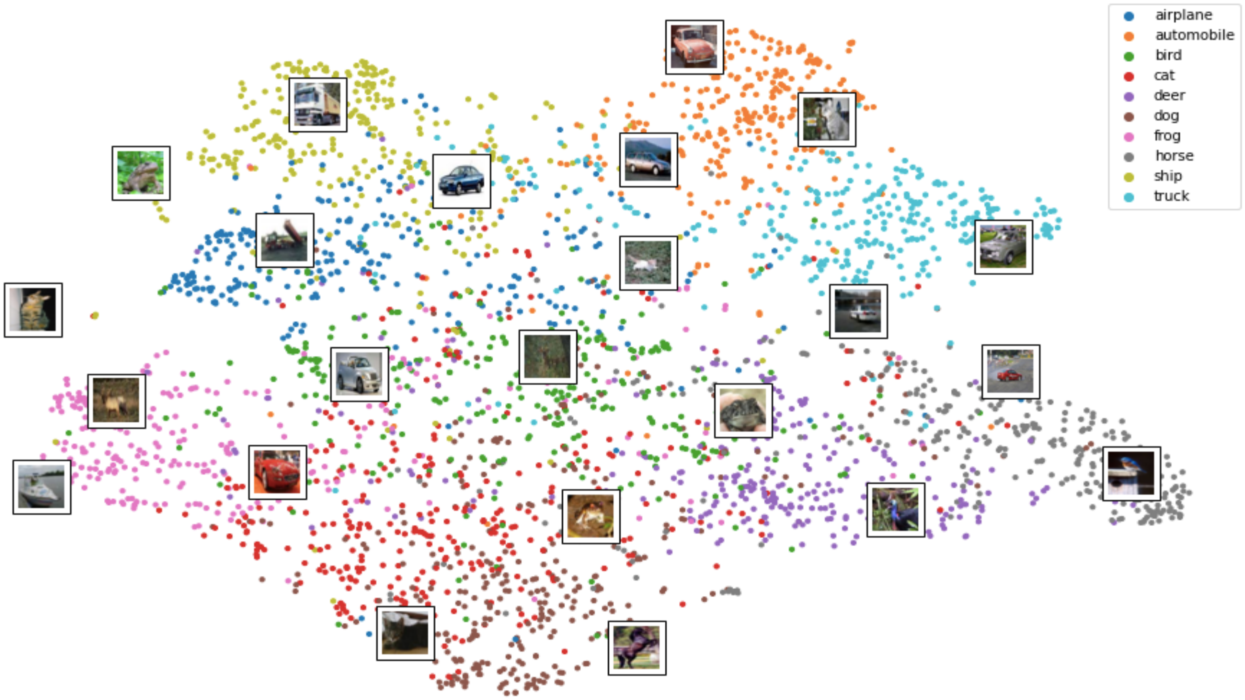
Fig.9 Worst DNN Model Confusion Matrix

We evaluated the model using confusion matrix to test the model, which depicts instances of a class being misclassified. When we compared the confusion matrix for the best model with the worst model (fig. 8 and fig.9), we see that a successful model prediction would illuminate the values on the matrix's diagonal line as seen in figure 8 which is CNN model. The second matrix shows that the predicted labels did not match the true labels in many instances which is confusion matrix for DNN model. None of the DNN model scored over 50% accuracy on the test data and validation data. The training loss and validation loss curve showed that the models are either overfitted or underfitted model in most cases. Hence, it proves that CNN modes are well-suited for CIFAR-10 dataset for image data with 3-dimensional input and output.

Finally, we can visualize the scatterplot shown below in figure 10 with color-coded pattern segmentation labels and images using t-Distributed Stochastic Neighbor Embedding (t-

SNE) which is another technique for dimensionality reduction and is particularly well suited for high-dimensional datasets. Using this technique, we reduced **Experiment 12** model's activation features hidden nodes from 256 to 2 with training on only most important input features.

t-Distributed Stochastic Neighbor Embedding technique – Experiment 12 (Best Model)



(Fig.10)

The accuracy of our best recommended model from **Experiment 12 is 75.14%** which is depicted in the scatter plot above using t-Distributed Stochastic Neighbor Embedding technique ten clusters on and non-overlapping mostly decent. There is a lot of overlap, most of them clearly segmenting into classes with overlapping color code of clusters this helps us in evaluating the model. Precision and recall are critical evaluation criteria. The precision of the classifier confidence is the fraction of recovered relevant instances, whereas recall, sensitivity, or true positive rate represents the prediction's true positive rate. Both values range between 0 and 1, with 1 indicating greater prediction. The F-measure is a classification accuracy metric which considers both the accuracy and recall values it displays the harmonic mean of these two values.

Conclusion

Knowing the benefits and limits of CNN is critical for maximizing its potential in real-world applications like as diagnostic imaging, with the objective of enhancing healthcare professionals' performance and improving patient care. CNN is substantially easier to train than other deep learning algorithms and can detect crucial characteristics with high accuracy without any human supervision. The CNN's powerhouse is the convolutional operation, which is responsible for constructing a reduced representation of the initial image while maintaining all its properties and removing irrelevant noise. One of the benefits of CNN is that the first convolutional layer is not connected to every pixel in the input image, which would be computationally inefficient, but just to the pixels in the receptive fields. While a Dense layer learns global patterns from all pixels in its input feature space.

Hence, I would recommend the management to use experiment 12 CNN model with 3 convolution layers 64, 128 and 128 nodes that has dense layers of 256 nodes following L2 regularize on dense layer only, 20 % dropout, batch normalization, and early stopping regularization techniques were used in the process of prevent model overfitting as a base case architecture to train using the CIFAR-10 data. Since it has an accuracy of 75.14% on test dataset which leads to quicker training time of 68 seconds due to less 357,642 parameters compared to other models and low-complexity model fits well.

In the future, I hope to train a convolutional neural network with less hidden nodes, which would reduce training time, model complexity, and trainable parameters. By evaluating convolutional layer further by tweaking hypermeters and experimenting with various regularization techniques to gather further knowledge on having decent accuracy compared to our best optimal model that is currently recommended to the management.

References

Chollet, F. 2018. *Deep Learning with Python*. Shelter Island, N.Y.: Manning. [ISBN-13: 978-1617294433]

Demush, Rostyslav. 2019. A Brief History of Computer Vision (and Convolutional Neural Networks). Retrieved from https://hackernoon.com/a-brief-history-of-computer-vision-and-convolutional-neural-networks-8fe8aacc79f3

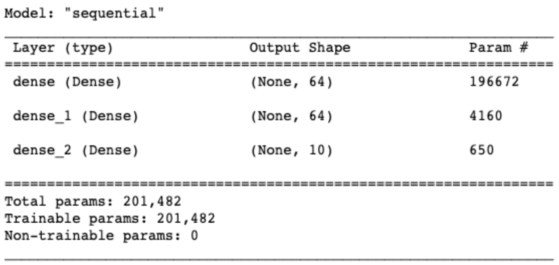
Kumar, Aayush, Ayush R Tripathi, Suresh Chandra Satapathy, and Yu-Dong Zhang. 2021 "SARS-Net: COVID-19 Detection from Chest x-Rays by Combining Graph Convolutional Network and Convolutional Neural Network.". Retrieved from https://www.sciencedirect.com/science/article/pii/S0031320321004350#cebibl

Sultana, Sufian, A., & Dutta, P. 2020. Evolution of Image Segmentation using Deep Convolutional Neural Network: A Survey. *Knowledge-Based Systems*, *201-202*, 106062–. https://doi.org/10.1016/j.knosys.2020.106062

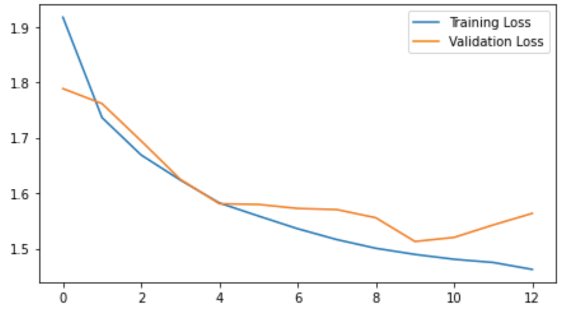
Yao, Wang, X., Wang, S.-H., & Zhang, Y.-D. 2020. A comprehensive survey on convolutional neural network in medical image analysis. *Multimedia Tools and Applications*. https://doi.org/10.1007/s11042-020-09634-7

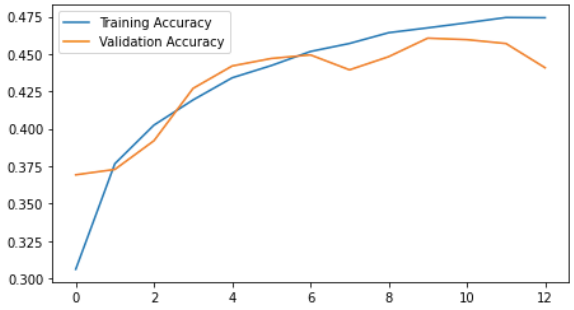
Appendix

Experiment 1: DNN 2 Layers with No Reg (Hidden Nodes: (64, 64) Dense Layer: 10)

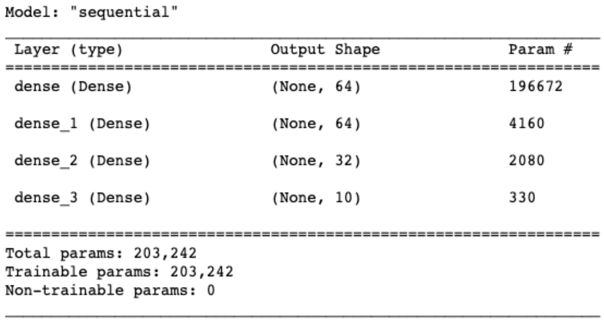


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                196672

 dense_1 (Dense)             (None, 64)                4160

 dense_2 (Dense)             (None, 10)                650

=================================================================
Total params: 201,482
Trainable params: 201,482
Non-trainable params: 0
_________________________________________________________________
```

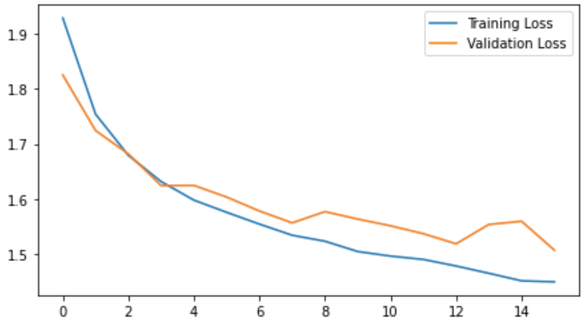


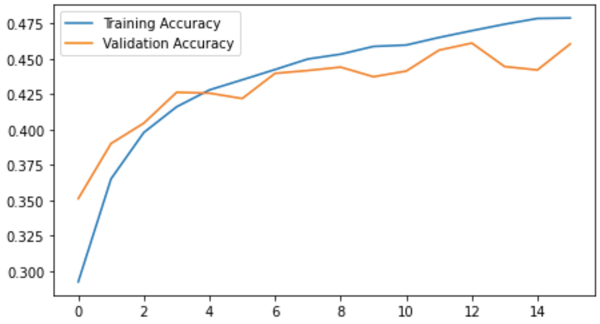


Experiment 2: DNN 3 Layers with No Reg (Hidden Nodes: (64,64,32) Dense Layer: 10)

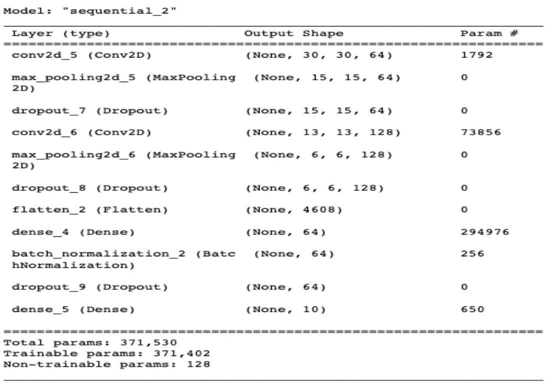


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                196672

 dense_1 (Dense)             (None, 64)                4160

 dense_2 (Dense)             (None, 32)                2080

 dense_3 (Dense)             (None, 10)                330

=================================================================
Total params: 203,242
Trainable params: 203,242
Non-trainable params: 0
_________________________________________________________________
```

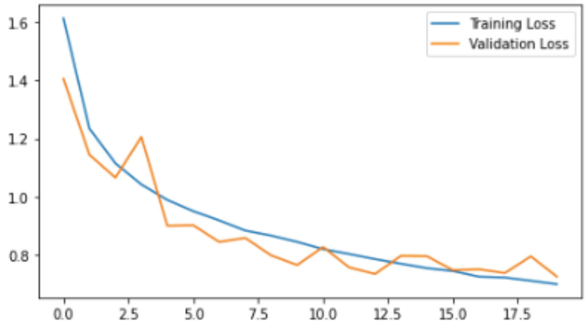


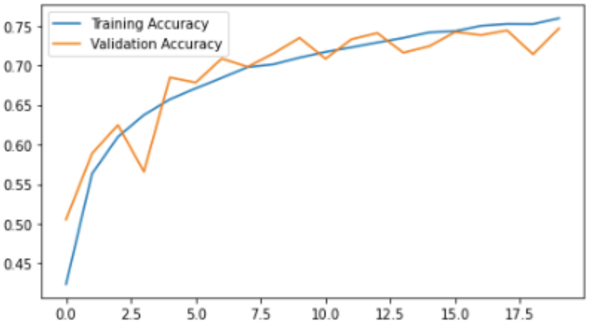


Experiment 3: CNN 2 Layers with No Reg (Hidden Nodes: (64,128), Dense Layer: (64,10))

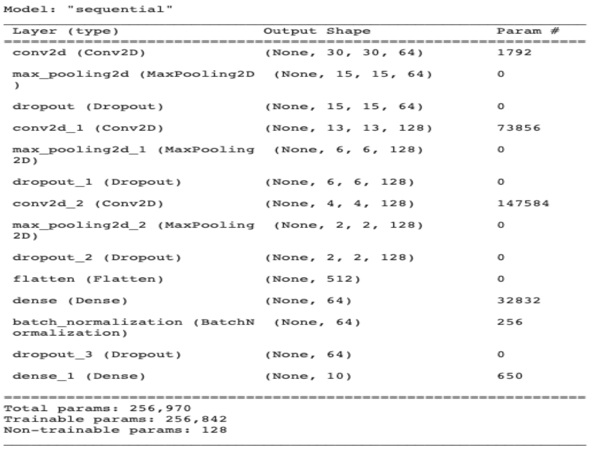


```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)           (None, 30, 30, 64)        1792

 max_pooling2d_5 (MaxPooling  (None, 15, 15, 64)       0
 2D)

 dropout_7 (Dropout)         (None, 15, 15, 64)        0

 conv2d_6 (Conv2D)           (None, 13, 13, 128)       73856

 max_pooling2d_6 (MaxPooling  (None, 6, 6, 128)        0
 2D)

 dropout_8 (Dropout)         (None, 6, 6, 128)         0

 flatten_2 (Flatten)         (None, 4608)              0

 dense_4 (Dense)             (None, 64)                294976

 batch_normalization_2 (Batc  (None, 64)               256
 hNormalization)

 dropout_9 (Dropout)         (None, 64)                0

 dense_5 (Dense)             (None, 10)                650

=================================================================
Total params: 371,530
Trainable params: 371,402
Non-trainable params: 128
_________________________________________________________________
```

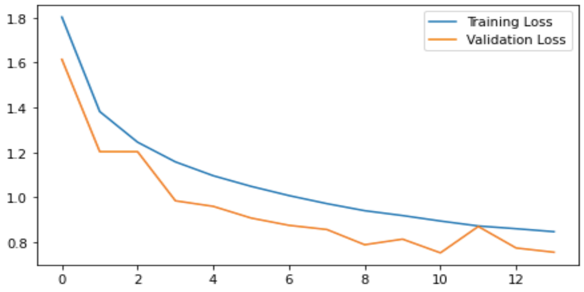


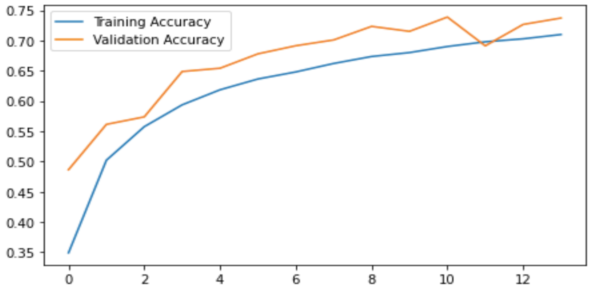


Experiment 4: CNN 3 Layers with No Reg (Hidden Nodes: (64,128, 128), Dense Layer: (64,10))

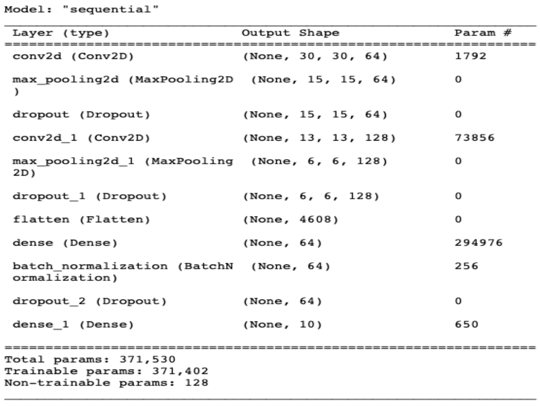


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 64)        1792

 max_pooling2d (MaxPooling2D  (None, 15, 15, 64)       0
 )

 dropout (Dropout)           (None, 15, 15, 64)        0

 conv2d_1 (Conv2D)           (None, 13, 13, 128)       73856

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 128)        0
 2D)

 dropout_1 (Dropout)         (None, 6, 6, 128)         0

 conv2d_2 (Conv2D)           (None, 4, 4, 128)         147584

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

 dropout_2 (Dropout)         (None, 2, 2, 128)         0

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 64)                32832

 batch_normalization (BatchN  (None, 64)               256
 ormalization)

 dropout_3 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 256,970
Trainable params: 256,842
Non-trainable params: 128
_________________________________________________________________
```

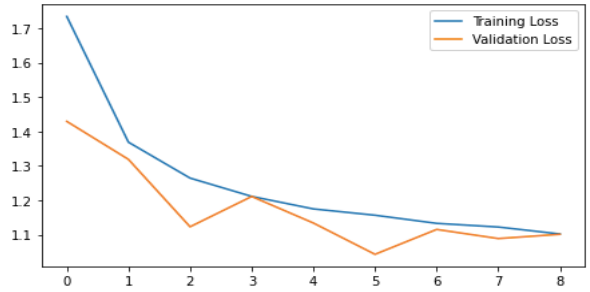


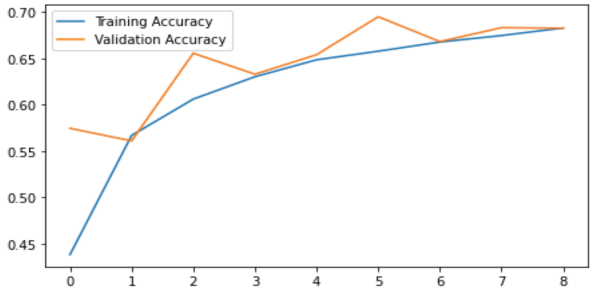


Experiment 5: CNN 2 Layers With Reg (Hidden Nodes: (64,128), Dense Layer: (64,10))

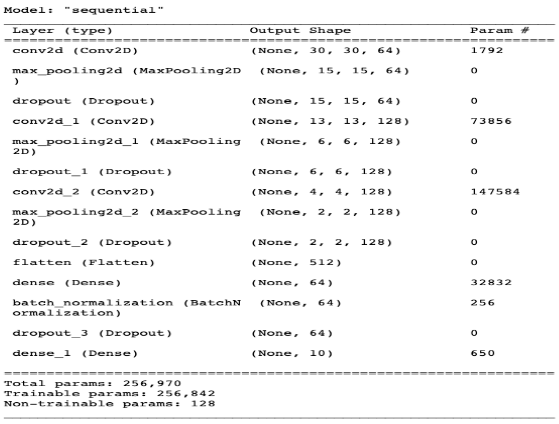


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 64)        1792

 max_pooling2d (MaxPooling2D  (None, 15, 15, 64)       0
 )

 dropout (Dropout)           (None, 15, 15, 64)        0

 conv2d_1 (Conv2D)           (None, 13, 13, 128)       73856

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 128)        0
 2D)

 dropout_1 (Dropout)         (None, 6, 6, 128)         0

 flatten (Flatten)           (None, 4608)              0

 dense (Dense)               (None, 64)                294976

 batch_normalization (BatchN  (None, 64)               256
 ormalization)

 dropout_2 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 371,530
Trainable params: 371,402
Non-trainable params: 128
_________________________________________________________________
```

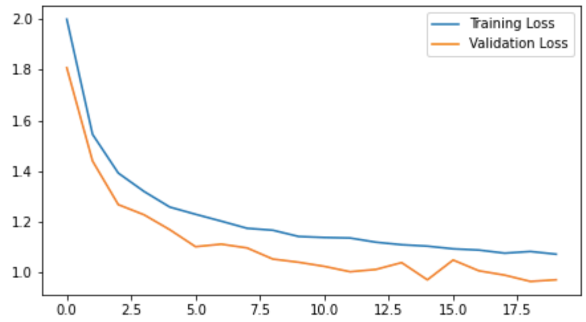


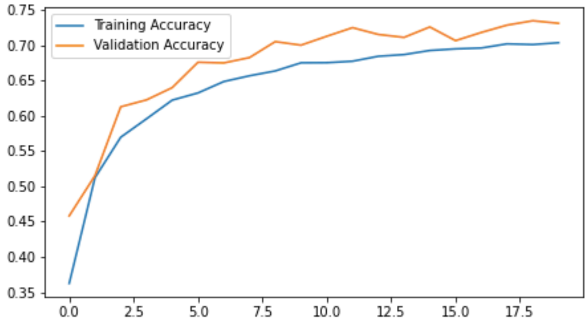


Experiment 6: CNN 3 Layers With Reg (Hidden Nodes: (64,128, 128), Dense Layer: (64,10))



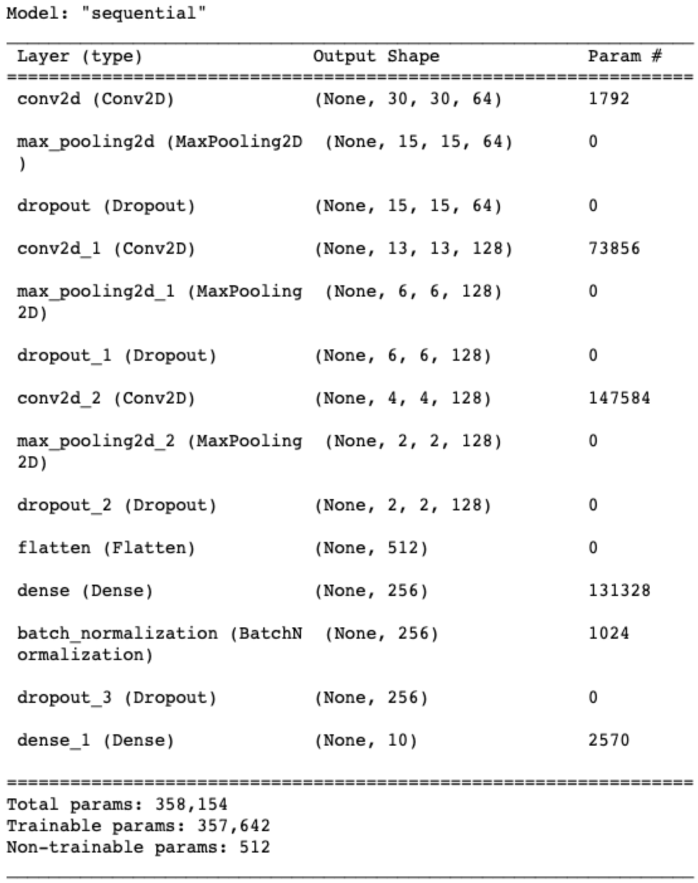
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 64)        1792

 max_pooling2d (MaxPooling2D  (None, 15, 15, 64)       0
 )

 dropout (Dropout)           (None, 15, 15, 64)        0

 conv2d_1 (Conv2D)           (None, 13, 13, 128)       73856

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 128)        0
 2D)

 dropout_1 (Dropout)         (None, 6, 6, 128)         0

 conv2d_2 (Conv2D)           (None, 4, 4, 128)         147584

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

 dropout_2 (Dropout)         (None, 2, 2, 128)         0

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 64)                32832

 batch_normalization (BatchN  (None, 64)               256
 ormalization)

 dropout_3 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 256,970
Trainable params: 256,842
Non-trainable params: 128
_________________________________________________________________
```



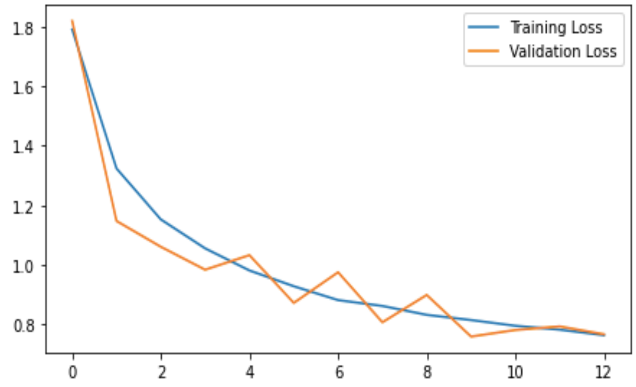


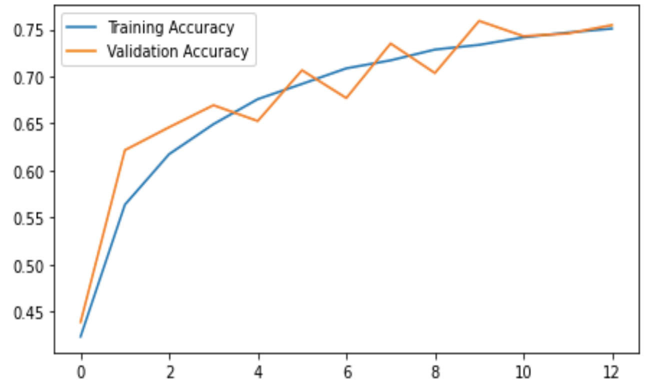
Experiment 12 (Final Best Model): CNN 3 Layer with (L2 Reg, Dropout 20%, Batch Norm)

(Hidden Nodes: (64,128, 128), Dense Layer: (256,10))



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 64)        1792

 max_pooling2d (MaxPooling2D  (None, 15, 15, 64)       0
 )

 dropout (Dropout)           (None, 15, 15, 64)        0

 conv2d_1 (Conv2D)           (None, 13, 13, 128)       73856

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 128)        0
 2D)

 dropout_1 (Dropout)         (None, 6, 6, 128)         0

 conv2d_2 (Conv2D)           (None, 4, 4, 128)         147584

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

 dropout_2 (Dropout)         (None, 2, 2, 128)         0

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 256)               131328

 batch_normalization (BatchN  (None, 256)              1024
 ormalization)

 dropout_3 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 10)                2570

=================================================================
Total params: 358,154
Trainable params: 357,642
Non-trainable params: 512
_________________________________________________________________
```


Training Loss
Validation Loss


Training Accuracy
Validation Accuracy