The “CIFAR-110”

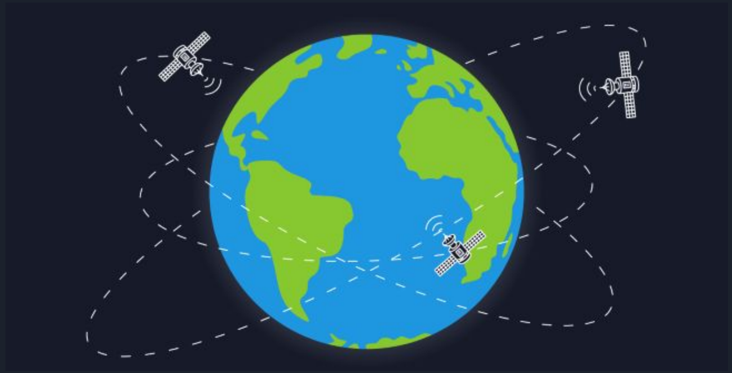
Image Classification Analysis via Convolutional Neural Networks

John A. Fonte, Esq.

Table of Contents

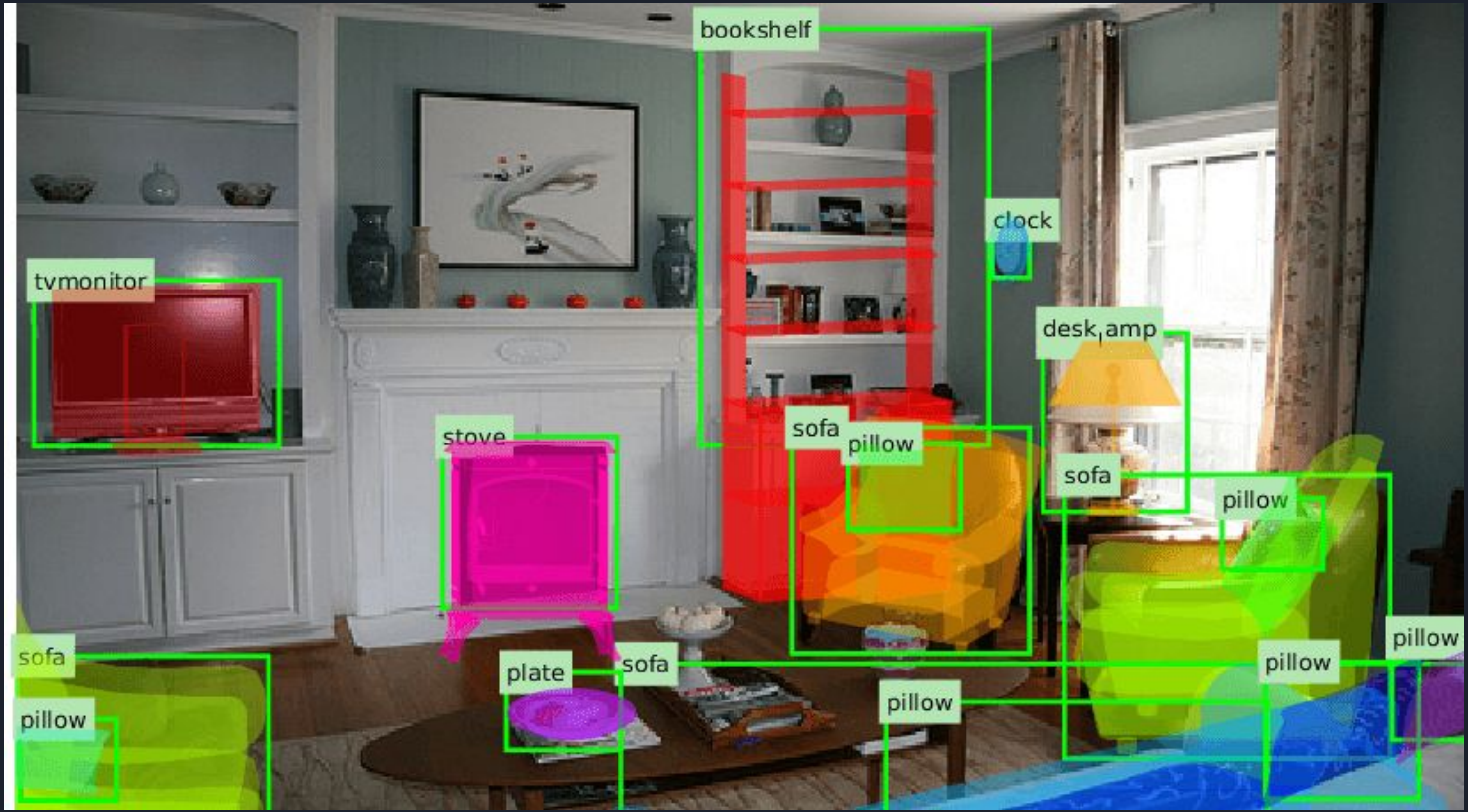
1. INTRODUCTION
a. The Problem & Its Significance

- Image Classification and Analysis has wide array of applications.
    - GPS and Geospatial Analysis
    - Facial Recognition
    - Radar and Depth Proximity (think auto-driving)



- And one more widespread application...

OBJECT RECOGNITION



Problems with Object Recognition

1. OR Models are Becoming More Sophisticated
    a. Identifying more kinds of objects
    b. Identifying more objects with higher specificity
        i. <u>Example:</u> Going from “Soda Can” to “Pepsi Can” or “Diet Coca-Cola” can
        ii. Identifying New Classes requires a re-training of the algorithm

2. OR Models are Highly Complex
    a. Almost always use Artificial Neural Networks
        i. Training Takes Forever
        ii. High Computational Resources are expended

Definition of Problem:

Which Object Recognition Model will Achieve the

Highest Efficiency?

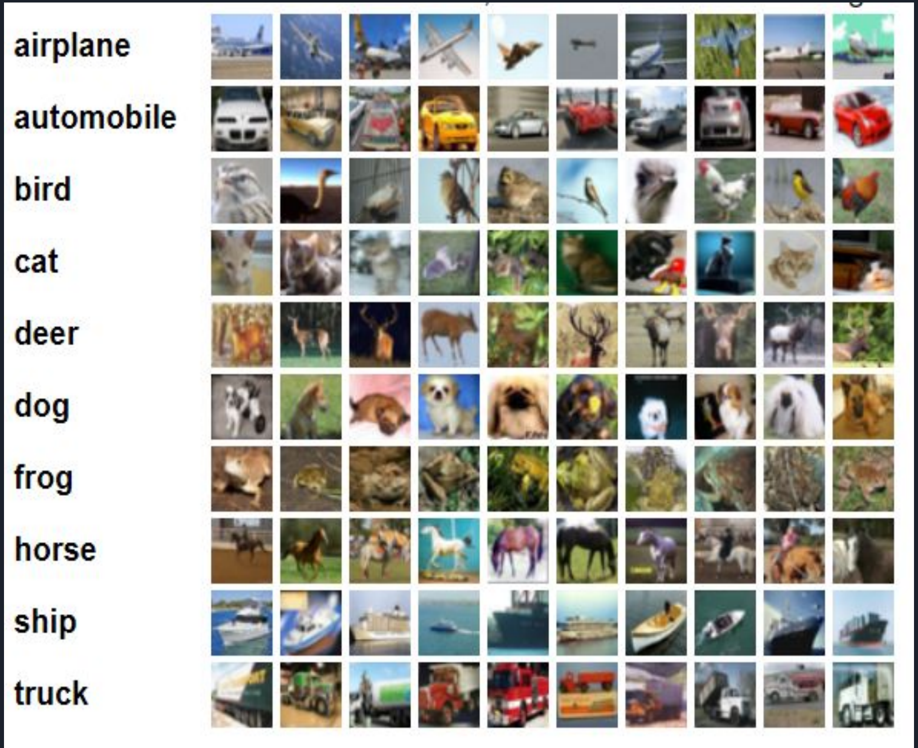
(Efficiency Defined Here: Accuracy-to-Time Rate)

Intended User: Company-side product, for companies to implement and adapt for their own user-base.

1. Introduction
b. Explanation of the Dataset: “CIFAR”

1. Canadian Institute For Advanced Research
   a. CIFAR-10
      i. 60000 images
         1. 50000 train, 10000 test
      ii. Balanced Classes
         1. Total of 6000 images / class
      iii. Uniform shape & Size
         1. 32 x 32 x 3(RGB channels)
         2. Typical 0 (darkest) to 255 (brightest) metric

   b. CIFAR-100
      i. Same Size (60000 images)
      ii. Balanced Classes
         1. 600 images / class
      iii. Uniform Shape & Size (32 x 32 x 3)



Datasets applied here: The "CIFAR-110"

- CIFAR-110 is CIFAR-10 with addition of CIFAR-10 classes
    - Additional classes = additional complexity
    - Switching it up from an otherwise commonly used image classification dataset
        - Both CIFAR-10 and CIFAR-110 can be loaded directly via

          ```
          from keras.datasets import cifar10
          ```

- Keeping CIFAR-10 for fast analysis and comparison of results to CIFAR-110

2. a. Data Loading

1. Steps:
   a. Downloaded compressed data files from CIFAR website
   b. Unzipped files
   c. Loaded files into Python's <u>Pickle</u> loading Library: and "unpickled" them
      i. Each file yielded a dictionary, with each dictionary key being a different set of array values

```
meta10 = unpickle('D:/Github/
meta10
```

```
{b'num_cases_per_batch': 10000,
 b'label_names': [b'airplane',
  b'automobile',
  b'bird',
  b'cat',
  b'deer',
  b'dog',
  b'frog',
  b'horse',
  b'ship',
  b'truck'],
 b'num_vis': 3072}
```

```
# checking the characteristics to import
print(batch100train.keys())
print(batch100test.keys())
```

```
dict_keys([b'filenames', b'batch_label', b'fine_labels', b'coarse_labels', b'data'])
dict_keys([b'filenames', b'batch_label', b'fine_labels', b'coarse_labels', b'data'])
```

2. Data Transformation
    a. Image Arrays Loaded as 1-D objects
    b. Need to Convert into Intended Shape of three dimensional (32x32x3)

```python
# Reshaping Input Array Data

def array_to_pixel_dimensionality_transposition(numarr):
    transposed_list = []

    for singlearray in numarr:
        singletransposed = singlearray.reshape(3,32,32).transpose([1, 2, 0])
        transposed_list.append(singletransposed)

    return transposed_list
```

3. Data Splitting: Already Split between Training and Testing for CIFAR-10 and -100
4. Creation of CIFAR-110
    a. Copy of random sample *of each class of equal size* in CIFAR-10 and append it to CIFAR-100, giving it new classification values
        i. Splitting those random samples proportionally to train and testing sets and append them accordingly.

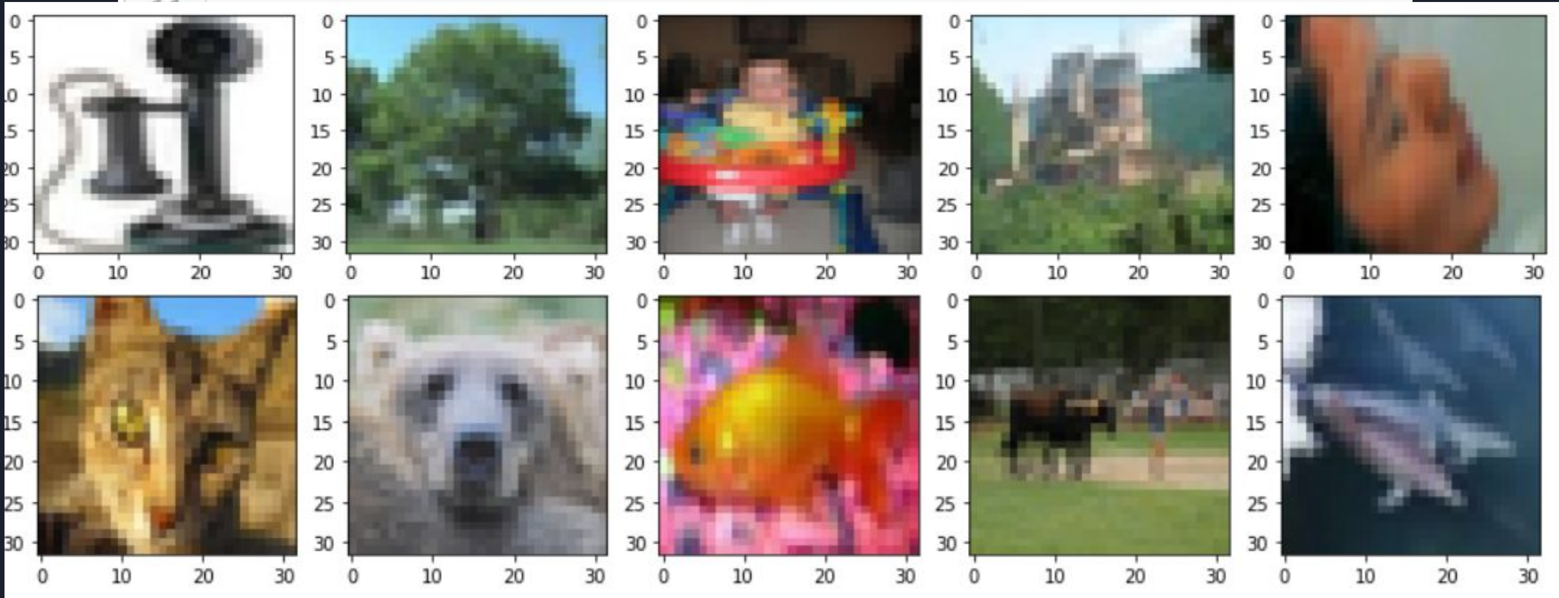
2. b. Data Splitting
- Creation of CIFAR-10 and CIFAR-110

1. Data is Already Split between Training and Testing, therefore CIFAR-10 is good to go
2. Creation of CIFAR-110
    a. Copy of random sample *of each class of equal size* in CIFAR-10 and append it to CIFAR-100, giving it new classification values
        i. Splitting those random samples proportionally to train and testing sets and append them accordingly.

3. Data Visualization

- Getting a visual of your dataset is always helpful!

```python
# sample images
plt.figure(figsize=(15,15))

sampleimages = df110train['Image Array'][600:610]

columns = 5
for i, image in enumerate(sampleimages):
    plt.subplot(len(sampleimages) / columns + 1, columns, i + 1)

    plt.imshow(image)
```



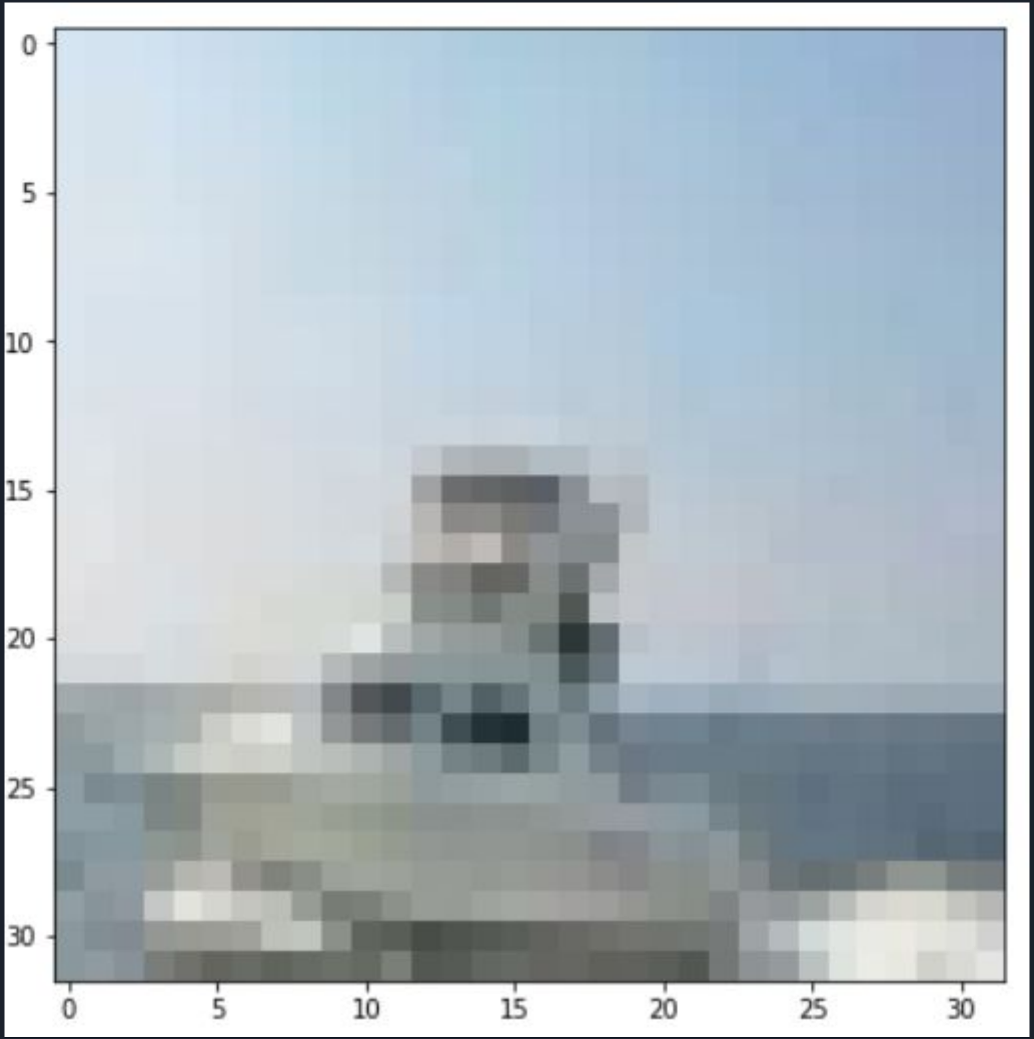
Final Changes

Low dimensionality (32 x 32 x 3) means difficulty in visualizing images BUT higher speed in terms of machine learning per image.

(fewer pixels means fewer features).

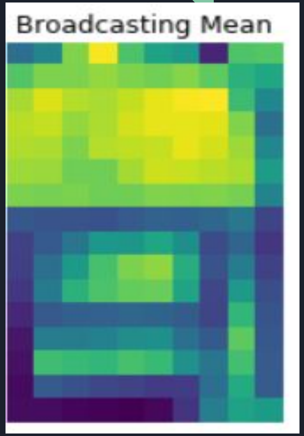
<u>HOWEVER:</u> Data is normalized down (scaling down from (0 to 255) to (0 to 1)

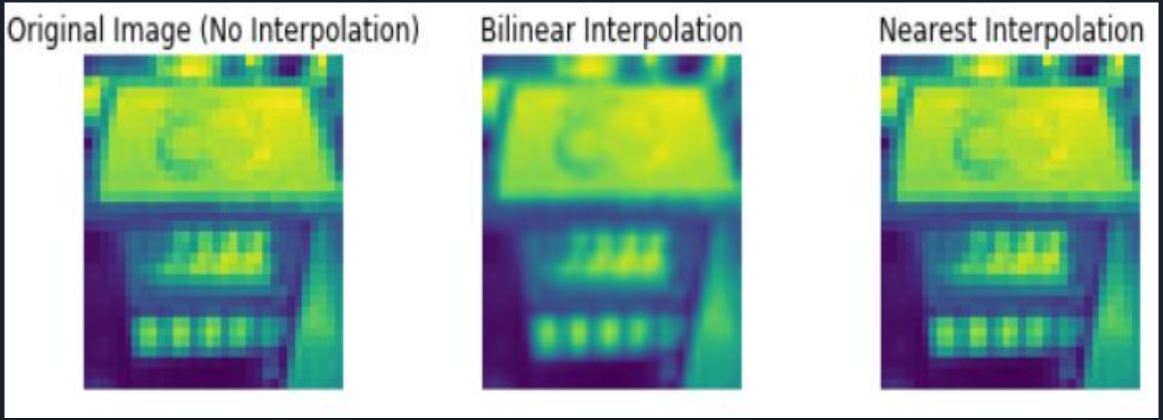
Picture of a Ship:

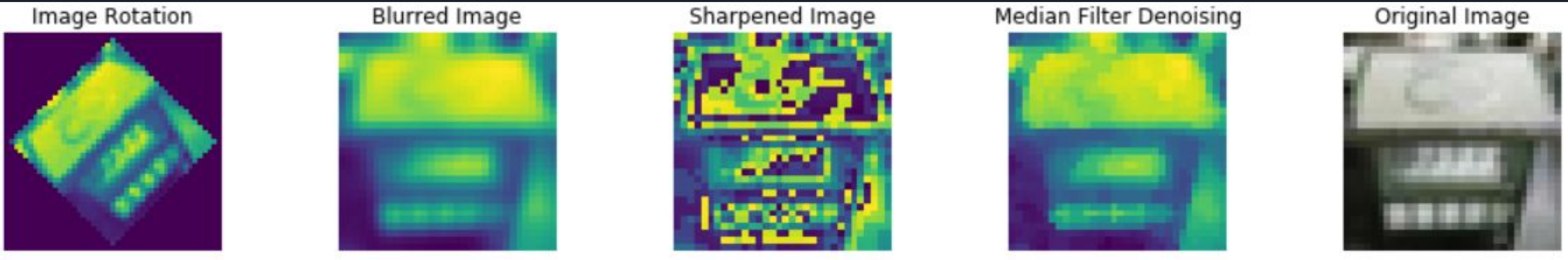


3. Data Manipulation

- Is there a Preprocessing Mechanism to obtain Sharper images?







4. Data Reduction Modeling

- Is it Possible to Increase Efficiency via Dimensionality Reduction?
  - Number of features per image = 32 x 32 x 3 = 3072
- Using a dimensionality reduction technique (e.g., PCA) would greatly increase efficiency, but can it be done on data requiring a spatial relationship?

- Using **Multi-Layer Perceptron (basic feed forward ANN) for baseline - no DR**

```
# only difference is that I dropped max_iter and n_iter_no_change - all else same

mlp = MLPClassifier(activation='relu', alpha=1e-05, batch_size='auto',
                    hidden_layer_sizes=(20,20,20), max_iter=150, random_state=444,
                    solver='adam', n_iter_no_change=6, verbose=False)
```

```
MLP Accuracy Score on testing set for CIFAR10 is: 43.0%

Time to run MLP model on CIFAR10 is 287 seconds
```

```
MLP Accuracy Score on testing set for CIFAR110 is: 1.0%

Time to run MLP model on CIFAR110 is 346 seconds
```

MLP applied to PCA-Reduced Data

- Determining Amount of Components

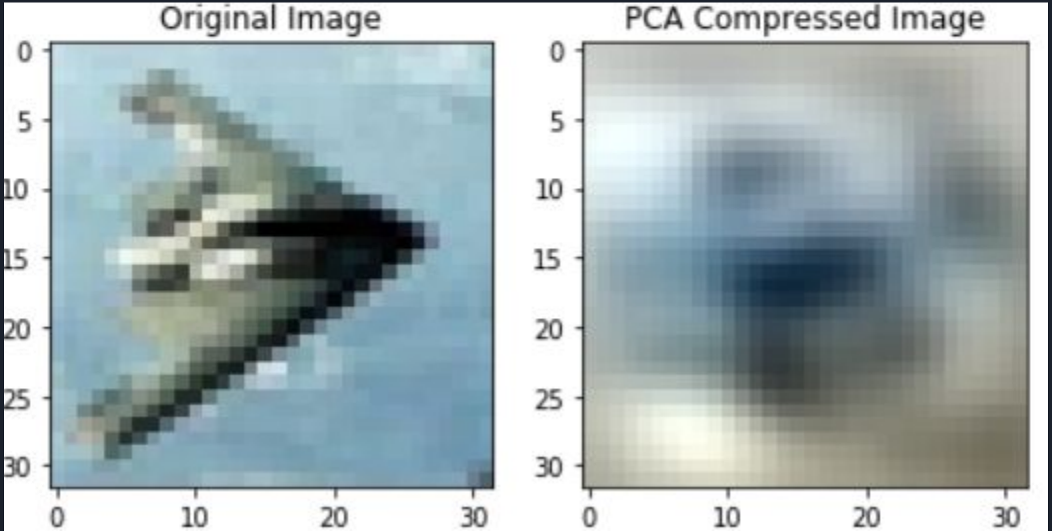
```
'''
We want to find the best tradeoff between lost variance and ease of computation,
and so we want to find the lowest number of dimensions that retains variance up to a designated amount
'''
# idea taken from Kaggle user Hamish Dickson for PCA analysis****************

def best_PCA_fit(fittingdata, minimumretainedvariance=0.857): # recommendation of amount of retained variance
                                                              # to be ~6 out of 7 items
    # model instantiation
    pcacheck = PCA()
    pcacheck.fit(fittingdata)
    cumulativesum = np.cumsum(pcacheck.explained_variance_ratio_)

    num_dimensions = np.argmax(cumulativesum >= minimumretainedvariance) + 1 # gotta have at least 1 dimension!

    print("The best number of dimensions to use for PCA on the fitting data is {}.".format(str(num_dimensions)))
```

```
The best number of dimensions to use for PCA on the fitting data is 59.
The best number of dimensions to use for PCA on the fitting data is 51.
```

MLP applied to PCA-Reduced Data

<u>Conclusion:</u>
DR Techniques eviscerate the image pixels' spatial relationship, thus significantly reducing accuracy despite also significantly reducing time.*

*<u>Note:</u> Model convergence was not achieved due to limiting max_iter hyperparameter.



MLP Testing...	CIFAR-10 Accuracy	CIFAR-10 Time	CIFAR-110 Accuracy	CIFAR-110 Time
Without PCA	43.0%	287 seconds	1.0%	346 seconds
With PCA	20.0%	32 seconds	1.0%	94 seconds

5. Unsupervised Modeling

- Unsupervised Models used:
  - Spectral Clustering
    - Performs pair-wise similarities between datapoints. This makes it good for non-parametric data such as image data, but pair-wise calculations are always slow.

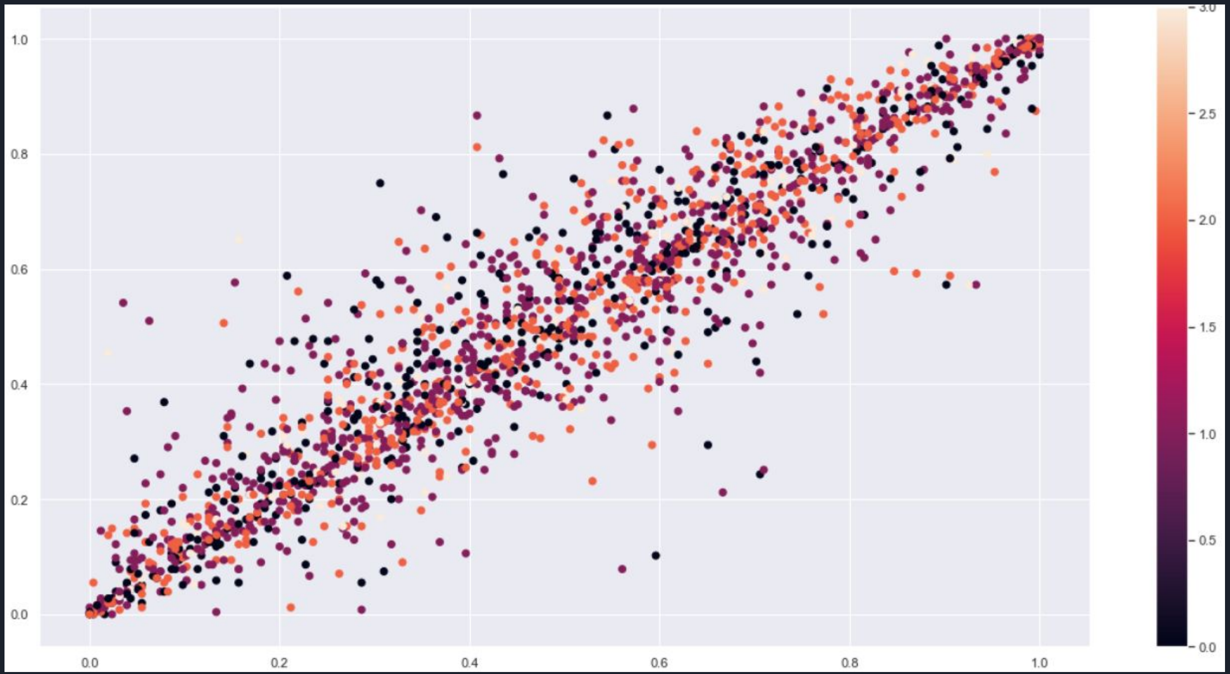
  - t-Stochastic Neighbor Embedding (t-SNE)
    - Stochastic gradient descent model that is also appropriate for non-parametric data Computationally heavy.

  - Linear Discriminant Analysis (LDA)
    - Creates a hyperplane finding the largest separation between classes (similar to SVM’s function in this respect).
      - Requires class input, thus making it not entirely unsupervised.

Spectral Clustering

CIFAR-10 Analysis

col_0	0	1	2	3
Target Labels				
0	58	80	61	13
1	49	88	58	10
2	41	91	57	15
3	41	78	52	8
4	41	79	63	9
5	31	80	66	7
6	59	87	54	24
7	40	89	61	15
8	34	86	57	19
9	49	70	63	17



Total time to run this model was: 3.4097113609313965 seconds.

Comparison between t-SNE and PCA

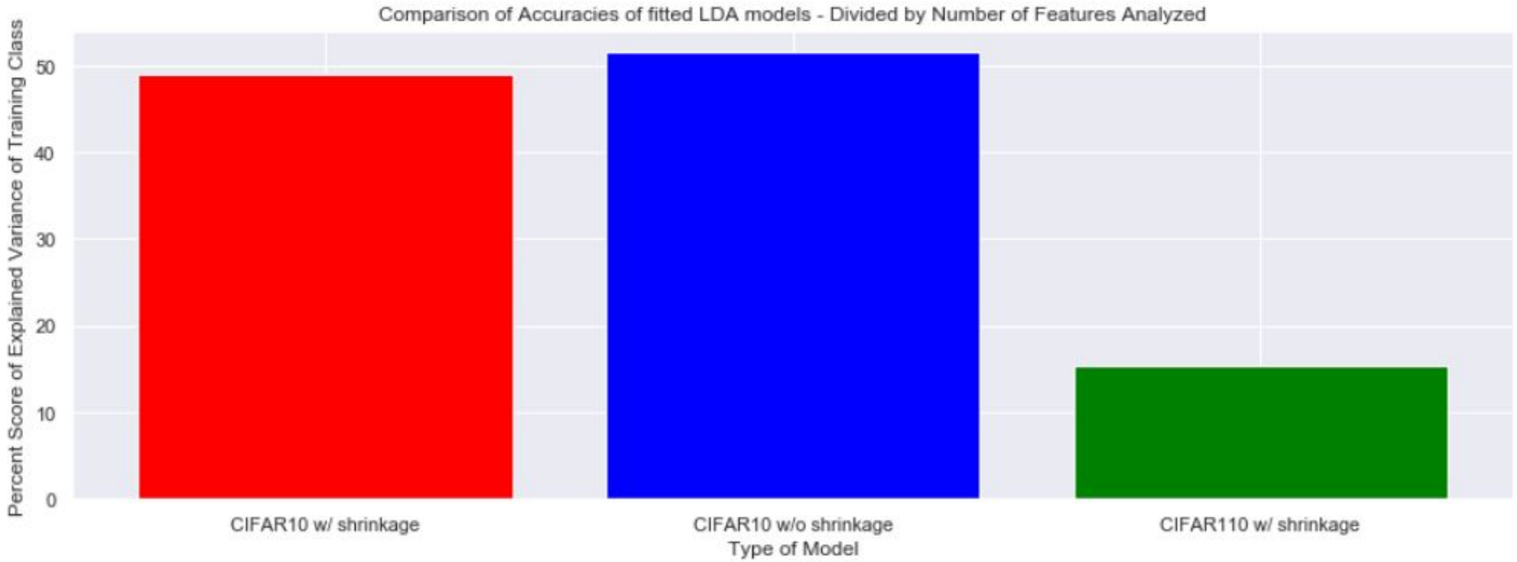
Conclusion: Despite the visualization seen earlier, simply graphing a PCA scatter yielded better unsupervised results than t-SNE.

PCA Conversion = ~9.3s
t-SNE analysis = 645s

Normally, t-SNE is the model that provides more spread-out results, but there was not full convergence after 300 iterations, taking the above amount of time.

LDA - Setup

```
LDA with shrinkage done! Time elapsed: 95.24511432647705 seconds
LDA without shrinkage done! Time elapsed: 37.49656271934509 seconds
LDA on CIFAR-110 with shrinkage done! Time elapsed: 142.13608598709106 seconds
```



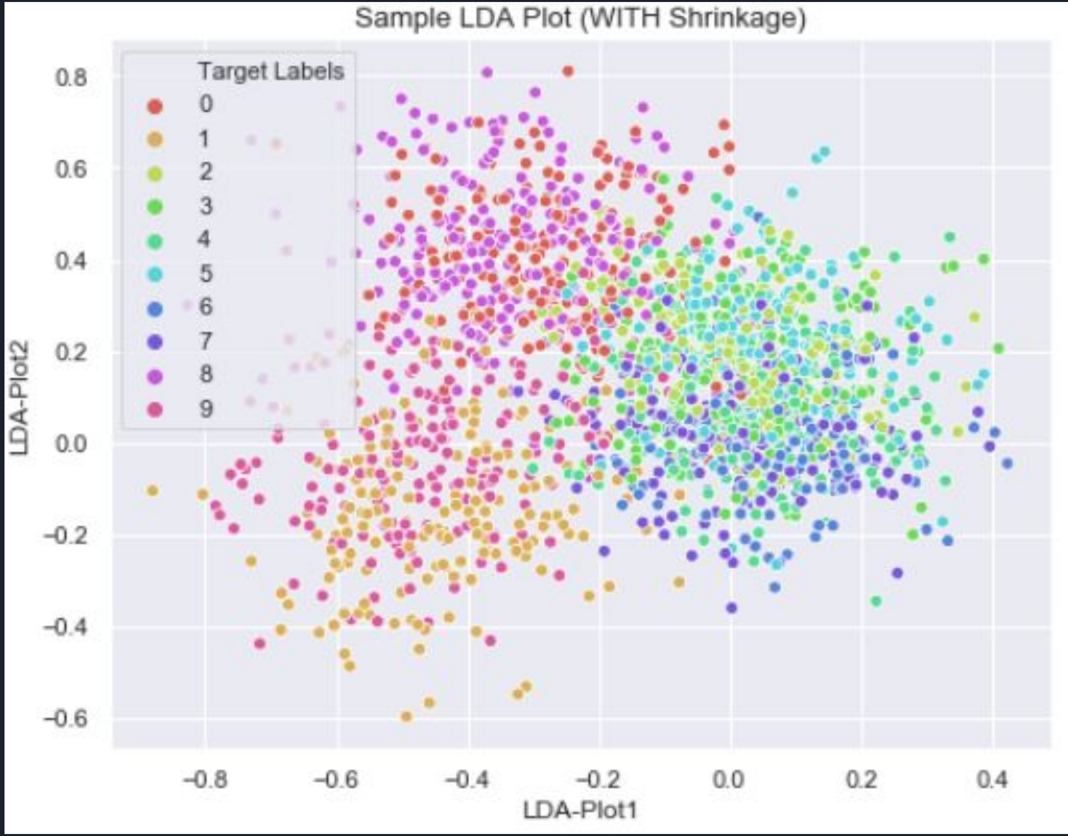
LDA
Count: 2000 points

<u>Conclusion:</u>

Expected clusters are found:

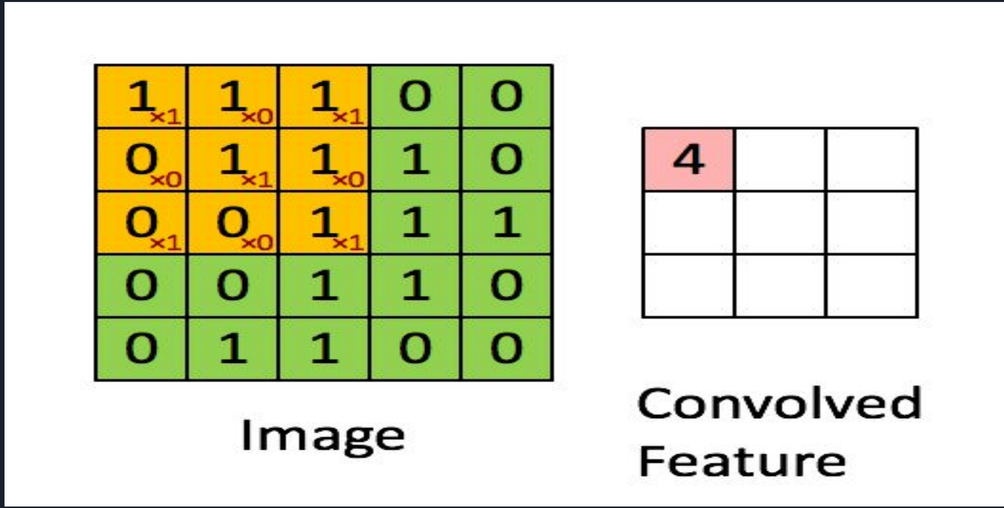
- 2,3,4(Bird,Cat,Deer)
- 5,7(Dog,Horse)
- 6 alone(Frog)
- 0,8(Airplane, Ship)
- 1,9 (Automobile, Truck)

This shows that there are no errant data trends.



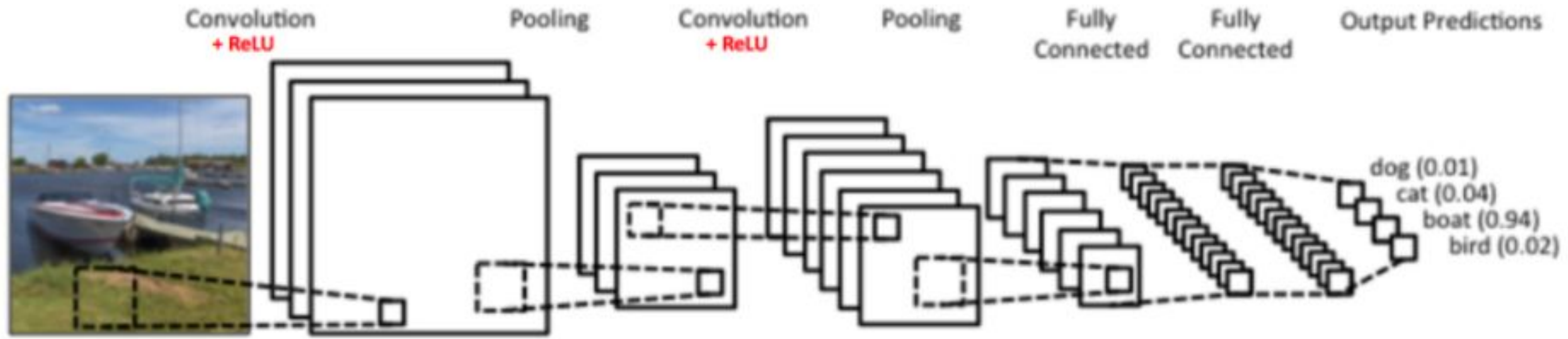
6. Supervised Learning
- Model of Choice: Convolutional Neural Networks (CNNs)

Why? Spatial relationship is taken into account based on its main characteristic: **CONVOLUTIONS**



Other machine learning models do not use this feature.

Overview of How CNNs Work:



STEPS TO CONVOLUTION

1. Define shape of input data
2. Define the convolutional planes (called ***kernels***)
3. Reduce the sample size into a pooling layer (a process called ***downsampling***)
4. Flatten the downsampled data, place flattened data into dense layers and run model

Overview of CNNs Used:

1. **Simple CNN**
   a. One convolutional layer

2. **Multi-Layer CNN**
   a. 3 convolutional layers of differing sizes, each with pooling layer immediately after, introduction of “padding”

3. **Small Layer CNN**
   a. 32-node layers, with 64 dense
   b. Introduction of BatchNormalization

4. **Combination CNN**
   a. Small layer-multi-CNN
   b. Contains padding and BatchNormalization

A. Simple CNN Architecture

```
# Building Model A.
def firstcnn(num_classes):
    model = Sequential()
    # First convolutional layer, note the specification of shape
    model.add(Conv2D(32, kernel_size=(3, 3),
                     activation='relu',
                     input_shape=inputshape))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.25))
    model.add(Dense(num_classes, activation='softmax'))

    model.compile(loss=categorical_crossentropy,
                  optimizer=adadelta,
                  metrics=['accuracy'])
    print(model.summary())
    return model
```

Layer (type)	Output Shape	Param #
conv2d_20 (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d_20 (MaxPooling	(None, 15, 15, 32)	0
dropout_23 (Dropout)	(None, 15, 15, 32)	0
flatten_12 (Flatten)	(None, 7200)	0
dense_23 (Dense)	(None, 128)	921728
dropout_24 (Dropout)	(None, 128)	0
dense_24 (Dense)	(None, 10)	1290

```
Total params: 923,914
Trainable params: 923,914
Non-trainable params: 0
```

For CIFAR-10. Total params for CIFAR-110 is 936,814.

Simple CNN Results

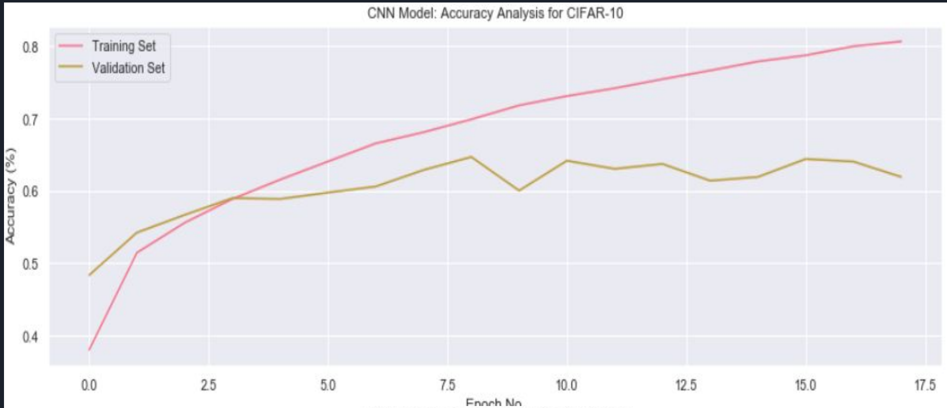
Accuracies for CIFAR-10 (top) and CIFAR-110 (bottom). Highly overfit!

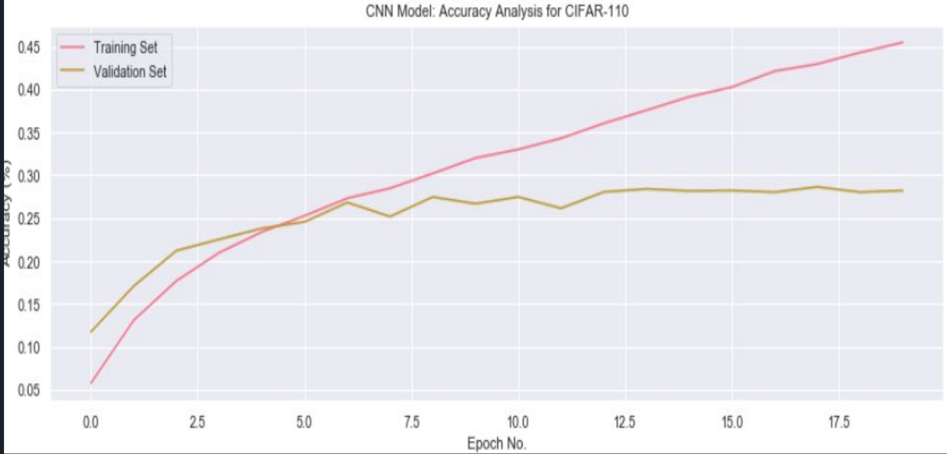
<u>Note:</u> All CNNs trained for 20 epochs

<u>Note 2:</u> Accuracy Graphs do not have uniform y-axis scaling.

<u>Note3:</u> No scaling done to results to place 10-class and 110-class datasets on equal footing.

Simple CNN Results	Accuracy on Testing Set	Total Time to Train
CIFAR-10	64%	806s
CIFAR-110	28%	856s





B. Multi-Layer CNN Architecture

```python
def multipleconvcnn(num_classes):
    model = Sequential()
    model.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=inputshape,padding='same'))
    model.add(MaxPooling2D((2, 2),padding='same'))
    model.add(Dropout(0.25))
    model.add(Conv2D(64, (3, 3), activation='relu',padding='same'))
    model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
    model.add(Dropout(0.25))
    model.add(Conv2D(128, (3, 3), activation='relu',padding='same'))
    model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.25))
    model.add(Dense(num_classes, activation='softmax'))

    model.compile(loss=categorical_crossentropy,
                  optimizer=adadelta,
                  metrics=['accuracy'])

    print(model.summary())

    return model
```

Layer (type)	Output Shape	Param #
conv2d_24 (Conv2D)	(None, 32, 32, 32)	896
max_pooling2d_24 (MaxPooling	(None, 16, 16, 32)	0
dropout_31 (Dropout)	(None, 16, 16, 32)	0
conv2d_25 (Conv2D)	(None, 16, 16, 64)	18496
max_pooling2d_25 (MaxPooling	(None, 8, 8, 64)	0
dropout_32 (Dropout)	(None, 8, 8, 64)	0
conv2d_26 (Conv2D)	(None, 8, 8, 128)	73856
max_pooling2d_26 (MaxPooling	(None, 4, 4, 128)	0
dropout_33 (Dropout)	(None, 4, 4, 128)	0
flatten_16 (Flatten)	(None, 2048)	0
dense_31 (Dense)	(None, 128)	262272
dropout_34 (Dropout)	(None, 128)	0
dense_32 (Dense)	(None, 10)	1290

Total params: 356,810
Trainable params: 356,810
Non-trainable params: 0

Total params for CIFAR-110 is 369,710. (less than Simple CNN).

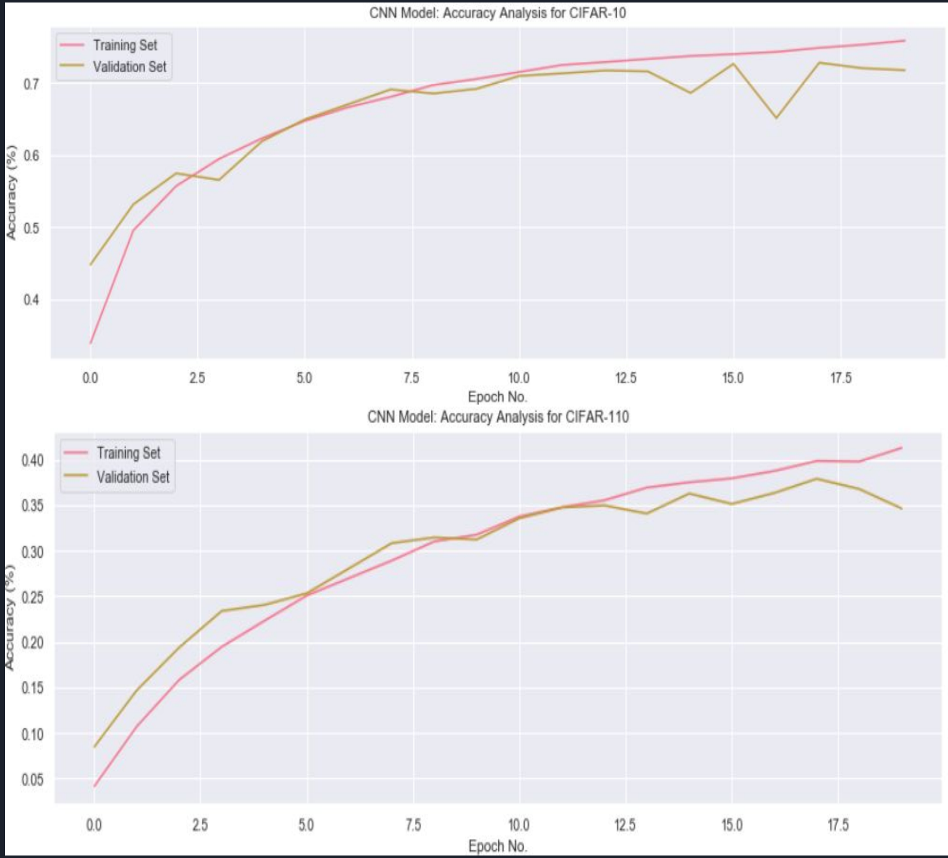
Multi-Layer CNN Results

padding='same' means that part of the kernel convolutes over *the outside of the image* for a uniformly-shaped convolution (essentially adding another row and column of pixels, each with "0".

Much less overfit than Simple CNN (although still slightly overfit).

Significantly more time to run than Simple CNN!

Multi-Layer CNN Results	Accuracy on Testing Set	Total Time to Train
CIFAR-10	72%	1170s
CIFAR-110	35%	1513s



C. Small-Layer CNN Architecture

```python
def small_layer_cnn(num_classes):
    model = Sequential()

    model.add(Conv2D(32, kernel_size=(3, 3),
                     activation='relu',
                     input_shape=inputshape))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.3))
    model.add(Conv2D(32, (3, 3), activation='relu',padding='same'))
    model.add(Conv2D(32, (3, 3), activation='relu',padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.3))
    model.add(Flatten())
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.4))
    model.add(Dense(num_classes, activation='softmax'))

    model.compile(loss=categorical_crossentropy,
                  optimizer='rmsprop',
                  metrics=['accuracy'])
    print(model.summary())
    return model
```

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_43 (Conv2D)           (None, 30, 30, 32)        896
_________________________________________________________________
batch_normalization_5 (Batch (None, 30, 30, 32)        128
_________________________________________________________________
max_pooling2d_43 (MaxPooling (None, 15, 15, 32)        0
_________________________________________________________________
dropout_59 (Dropout)         (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_44 (Conv2D)           (None, 15, 15, 32)        9248
_________________________________________________________________
conv2d_45 (Conv2D)           (None, 15, 15, 32)        9248
_________________________________________________________________
batch_normalization_6 (Batch (None, 15, 15, 32)        128
_________________________________________________________________
max_pooling2d_44 (MaxPooling (None, 7, 7, 32)          0
_________________________________________________________________
dropout_60 (Dropout)         (None, 7, 7, 32)          0
_________________________________________________________________
flatten_25 (Flatten)         (None, 1568)              0
_________________________________________________________________
dense_49 (Dense)             (None, 64)                100416
_________________________________________________________________
dropout_61 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_50 (Dense)             (None, 10)                650
=================================================================
Total params: 120,714
Trainable params: 120,586
```

Total params for CIFAR-110 is 127,714

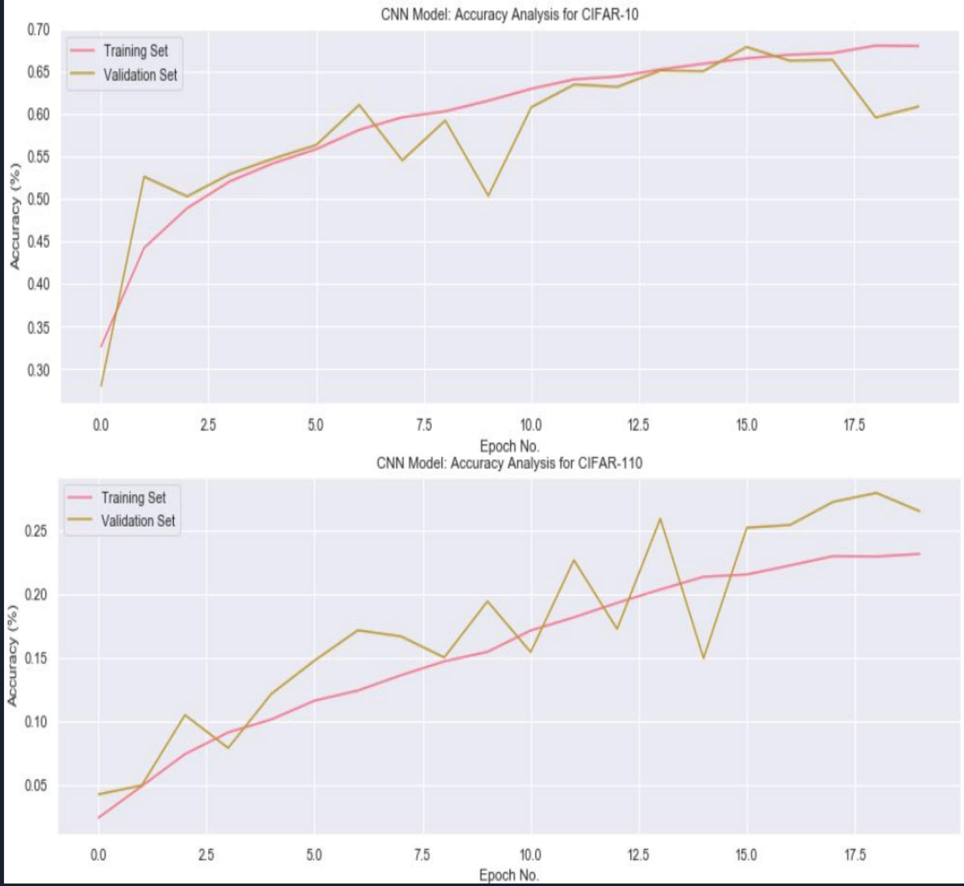
Small-Layer CNN Results

<u>BatchNormalization():</u> As the name implies, it initializes the parameters with zero mean and unit variance.

Sporadic validation accuracies are seen here. Perhaps the holdout (of 10% of the training set) is too small for consistent results, although this was not a problem for the other CNNs.

Training times increased to 40 and 50 minutes (using 100% CPU and several GB of RAM).

Small-Layer CNN Results	Accuracy on Testing Set	Total Time to Train
CIFAR-10	61.05%	2434s
CIFAR-110	26.22%	3102s



D. “Combination” CNN Architecture

```python
def combo_cnn(num_classes):
    model = Sequential()
    model.add(Conv2D(32, kernel_size=(4, 4),activation='relu',input_shape=inputshape,padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2),padding='same'))
    model.add(Dropout(0.3))
    model.add(Conv2D(32, (3, 3), activation='relu',padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
    model.add(Dropout(0.3))
    model.add(Conv2D(64, (3, 3), activation='relu',padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
    model.add(Dropout(0.4))

    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))

    model.compile(loss=categorical_crossentropy,
                  optimizer='adam',
                  metrics=['accuracy'])

    print(model.summary())

    return model
```

Layer (type)	Output Shape	Param #
conv2d_55 (Conv2D)	(None, 32, 32, 32)	1568
batch_normalization_15 (Batc	(None, 32, 32, 32)	128
max_pooling2d_53 (MaxPooling	(None, 16, 16, 32)	0
dropout_73 (Dropout)	(None, 16, 16, 32)	0
conv2d_56 (Conv2D)	(None, 16, 16, 32)	9248
batch_normalization_16 (Batc	(None, 16, 16, 32)	128
max_pooling2d_54 (MaxPooling	(None, 8, 8, 32)	0
dropout_74 (Dropout)	(None, 8, 8, 32)	0
conv2d_57 (Conv2D)	(None, 8, 8, 64)	18496
batch_normalization_17 (Batc	(None, 8, 8, 64)	256
max_pooling2d_55 (MaxPooling	(None, 4, 4, 64)	0
dropout_75 (Dropout)	(None, 4, 4, 64)	0
flatten_29 (Flatten)	(None, 1024)	0
dense_57 (Dense)	(None, 128)	131200
dropout_76 (Dropout)	(None, 128)	0
dense_58 (Dense)	(None, 10)	1290

Total params: 162,314
Trainable params: 162,058
Non-trainable params: 256

Total params for CIFAR-110 is 175,214.

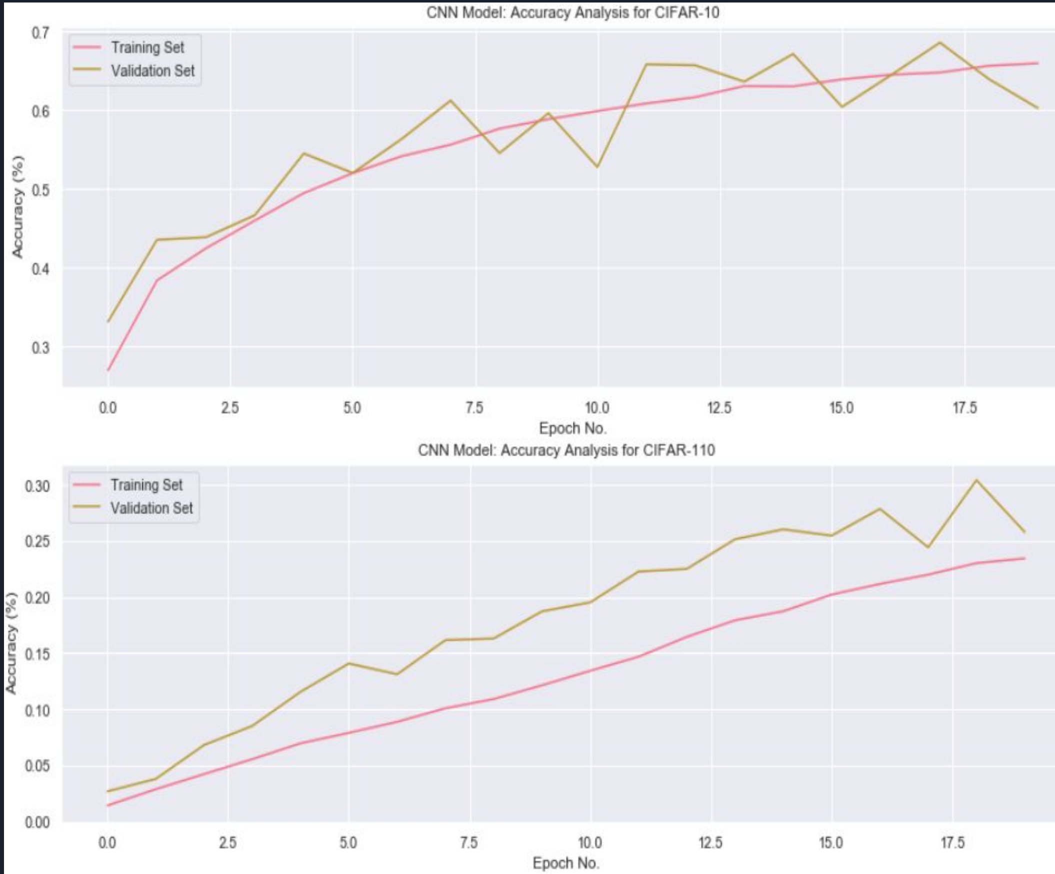
Combination CNN Results

Applied BatchNormalization, padding, and multiple small layers (i.e., the combination of Models A, B, & C).

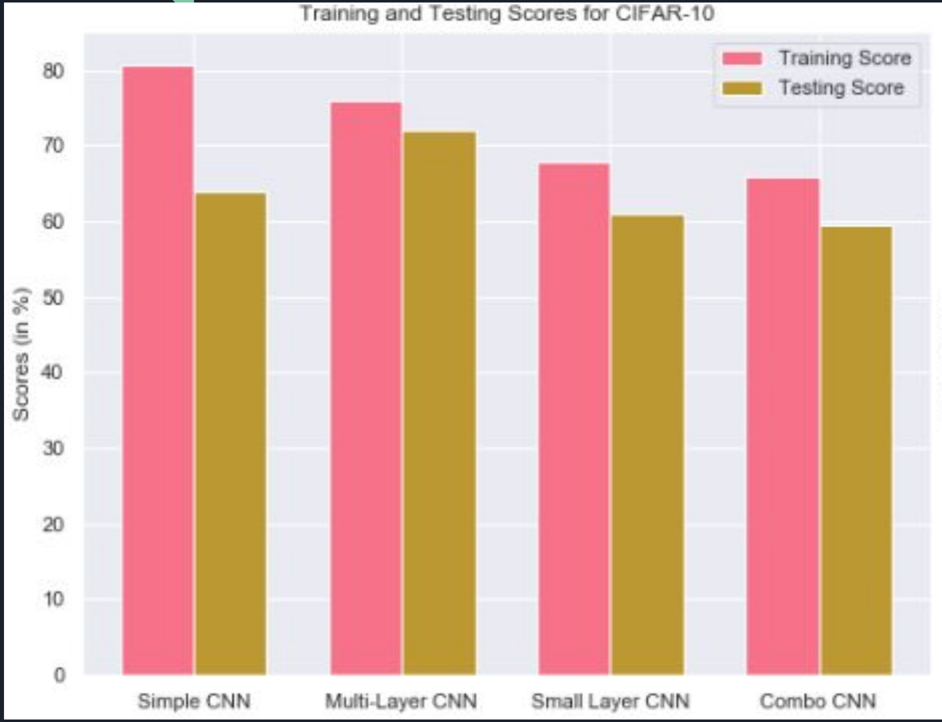
Least overfit of all the models, with CIFAR-110 training being *underfit.*

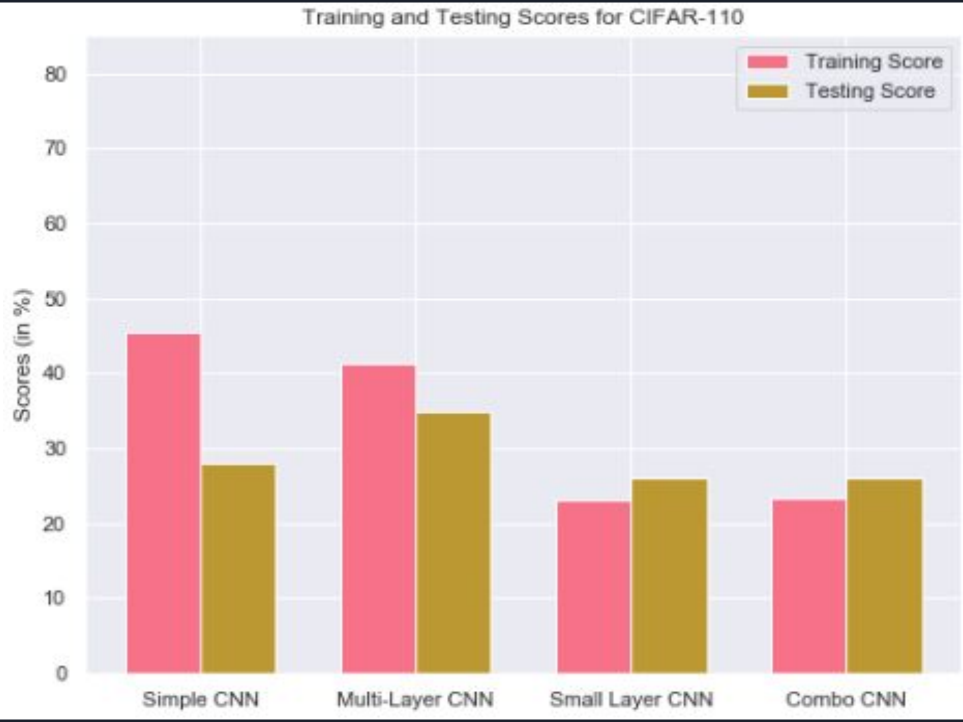
Despite normalizations meant to streamline/speed up training, this model took the longest time to train!

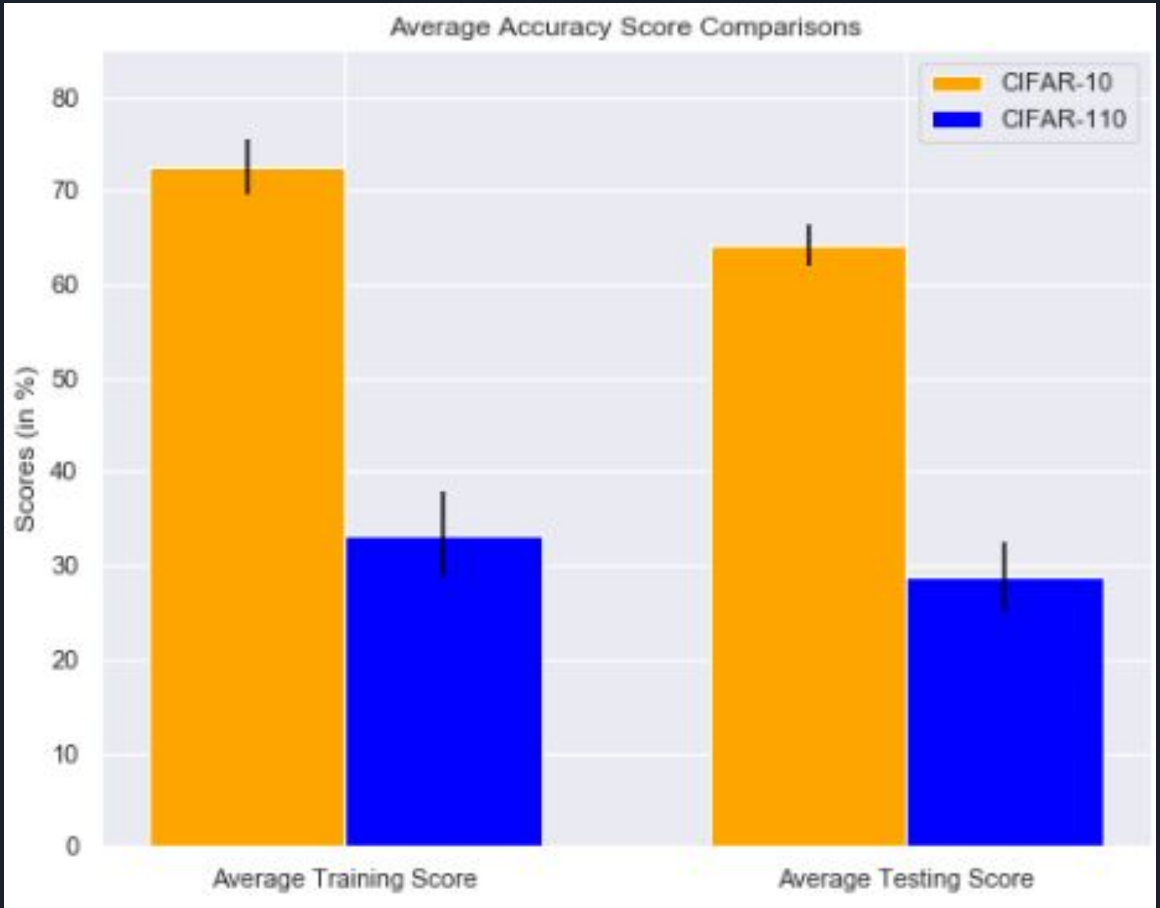
Combination CNN Results	Accuracy on Testing Set	Total Time to Train
CIFAR-10	59.55%	3032s
CIFAR-110	26.04%	3396s



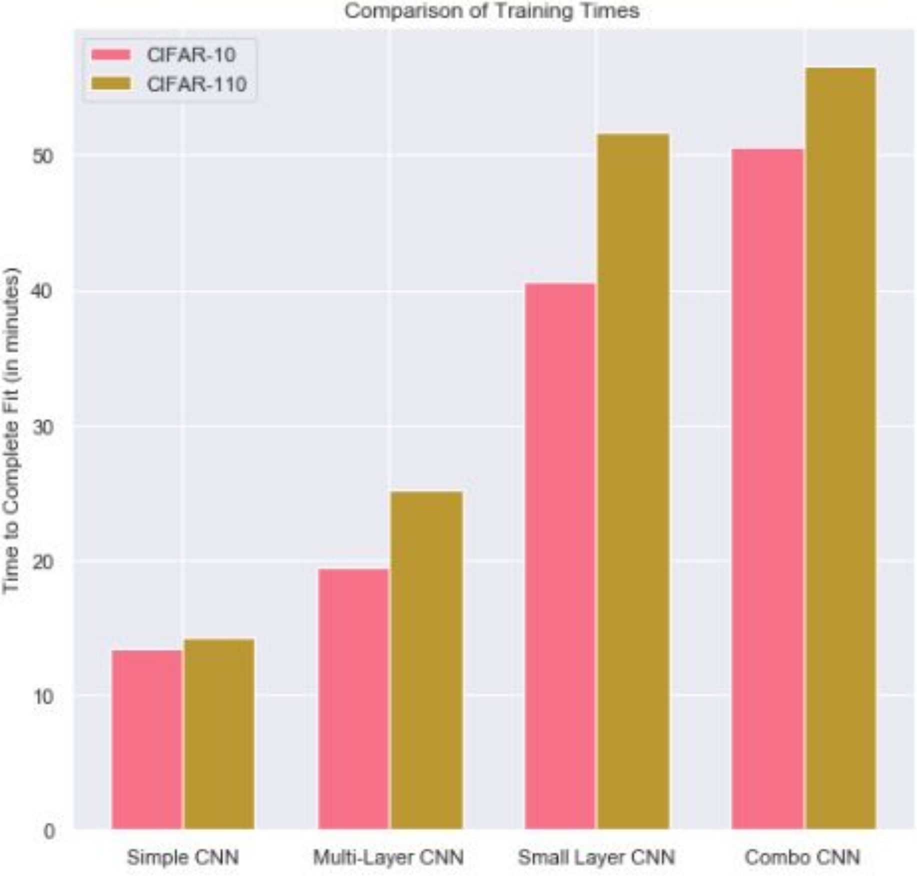
7. CONCLUSION - Accuracies

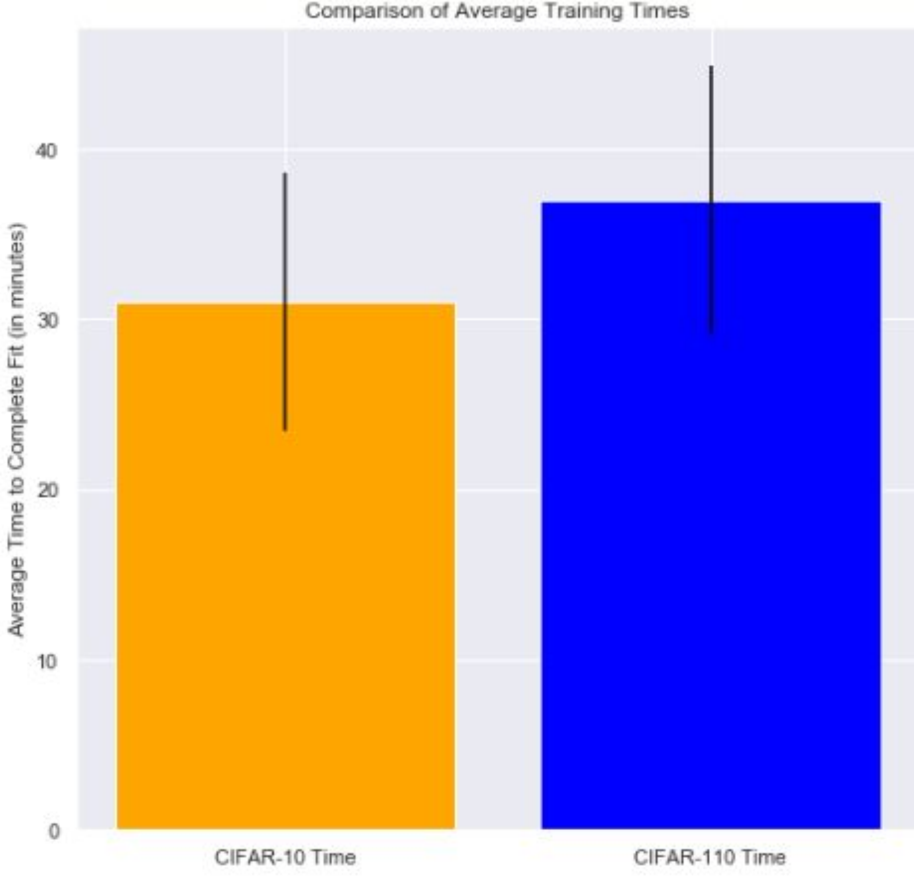





Average Accuracy Score Comparisons
CIFAR-10
CIFAR-110
Scores (in %)
0
10
20
30
40
50
60
70
80
Average Training Score
Average Testing Score

7. CONCLUSION - Training Times





7. CONCLUSION - Efficiency Scores (measured in Accuracy / Min. Rates)

7. CONCLUSION - FINAL ANALYSIS

The Problem: Which supervised model proves to be the best accuracy/efficiency tradeoff for image classification?

The Answer: Out of the four CNN models tested, the

Multi-Layer CNN Keras Model

is the best to use for both small classification sets (CIFAR-10) and large classification sets.

Explanation: The Simple CNN, as the name would suggest, converged the fastest. This gave it the *best efficiency score out of all models* for both CIFAR-10 and CIFAR-110 datasets. At the same time, the simple CNN **was also the most overfit of the four models.** The Multi-Layer CNN was the least overfit when applied to both CIFAR-10 and CIFAR-110 datasets. It also was the second fastest and thus had the second highest efficiency score.

F.A.Q.

1. **Why weren't other metrics used, including precision, specificity, or F1 scoring?**
   - **These metrics make more sense for a binary classifier: what exactly is a "false negative" for a multi-classification problem??? For this reason, accuracy was focused on (and its modelling analog, loss function calculation).**

2. **Why is training the focus here? Once a model is trained and sample weights are obtained for a given model, wouldn't making testing predictions be the point of focus for speed?**
   - **Maybe, but focusing on training time nonetheless speaks to this anyway. There is no good way to measure prediction time, since it happens so fast. The fractions of a second to make a prediction *should* be proportionate (not equal) to training time. If the focus of the project is efficiency, then having a much larger scale to analyze, i.e., training time, would be more appropriate than testing time.**

3. **What was the purpose of combining CIFAR-10 and CIFAR-100 to create CIFAR-110?**
   - **Much like MNIST and other popular image datasets - where the data can literally be called by a keras import - the CIFAR datasets have been analyzed *ad nauseum*. I wanted to explore how the introduction of new classes would affect CIFAR-110, especially when those new classes are relatively similar to some of those 100 classes. In short, this is an added challenge for modeling purposes to buttress the results of those models.**


Thank You for your attention.
Any questions?
(this is the part where you run)
someecards
user card