**Capstone Project**
Machine Learning Engineer Nanodegree

**Ananthesh J Shet**
31-Dec-2017

# Object Classification Using Convolutional Neural Network for Image Search

## I. Definition

### Project Overview

There are a lot of images and videos out there in the wild and they can be used for a wide variety of applications. Users search for images for entertainment, academics, presentations, projects etc. Users also need a large number of images belonging to specific category for training applications[1] like Autonomous Cars, Gesture Recognition, Optical Character Recognition, Face Recognition, Remote Sensing, Machine Vision, Robots etc. Users search for these images on the net and it is important to get the right images for the right applications. For a good user experience and for faster search, search engines have to provide a bunch of relevant images based on certain keywords in less time.

Relevant images are those images which have in them the object(s) mentioned in the keyword(s). For e.g., if the user searches for images by entering keyword "car", then any image having car in it, maybe among other objects, is a relevant image. So the search engine has to first classify objects in the images and return those images containing objects of interest.

In [1], Sandeep Kumar et al. discusses different machine learning techniques such as Decision Tree, Support Vector Machine and KNN-Classification for image classification. In [2], Er. Navjot Kaur et al. use SVM RBF Kernel for Object classification. In [3], Jost Tobias Springenberg et al. make use of only Convolution Neural Network(CNN) for object classification.

### Problem Statement

Classifying objects in an image is very important for a successful image search. Classifying objects incorrectly will lead to unwanted images in the search result and a frustrated user! A general classifying algorithm takes a set of features characterizing an object and uses them to

[1] https://en.wikipedia.org/wiki/Category:Applications_of_computer_vision

determine the object class. Some algorithms need the user to specify the features whereas some algorithms can learn the features themselves.

A search engine fails if the classifying algorithm performs poorly. If the algorithm recognizes an object when it is not present, then it is termed as False Positive(FP). If the algorithm doesn't recognize the object when it is really present, then it is termed as False Negative(FN). High FP or FN indicate a poorly performing algorithm. This project aims at designing a multi-class classification system using simple architecture CNN model with low FP and FN. The input image has to be pre-processed to help the model to start with. A CNN has to be designed by making use of different neural network layers. Then the final model has to classify an input image into one of the given N classes. The type of the input image and the different classes are discussed in the Data Exploration section below.

# II. Analysis

## Data Exploration

For this project, I used the publicly available dataset from CIFAR-100[2]. The CIFAR-100 dataset consists of 60000 32x32 color images in 100 classes with 600 images per class. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 super classes. Each image comes with a "fine" label (the class to which it belongs) and a "coarse" label (the super class to which it belongs).

Here is the list of classes in the CIFAR-100:

| # | Superclass | Classes |
|---|---|---|
| **0** | aquatic mammals | beaver, dolphin, otter, seal, whale |
| **1** | fish | aquarium fish, flatfish, ray, shark, trout |
| **2** | flowers | orchids, poppies, roses, sunflowers, tulips |
| **3** | food containers | bottles, bowls, cans, cups, plates |
| **4** | fruit and vegetables | apples, mushrooms, oranges, pears, sweet peppers |
| **5** | household electrical devices | clock, computer keyboard, lamp, telephone, television |
| **6** | household furniture | bed, chair, couch, table, wardrobe |
| **7** | insects | bee, beetle, butterfly, caterpillar, cockroach |
| **8** | large carnivores | bear, leopard, lion, tiger, wolf |
| **9** | large man-made outdoor things | bridge, castle, house, road, skyscraper |
| **10** | large natural outdoor scenes | cloud, forest, mountain, plain, sea |
| **11** | large omnivores and herbivores | camel, cattle, chimpanzee, elephant, kangaroo |

[2] https://www.cs.toronto.edu/~kriz/cifar.html

| | | |
|---|---|---|
| **12** | medium-sized mammals | fox, porcupine, possum, raccoon, skunk |
| **13** | non-insect invertebrates | crab, lobster, snail, spider, worm |
| **14** | people | baby, boy, girl, man, woman |
| **15** | reptiles | crocodile, dinosaur, lizard, snake, turtle |
| **16** | small mammals | hamster, mouse, rabbit, shrew, squirrel |
| **17** | trees | maple, oak, palm, pine, willow |
| **18** | vehicles 1 | bicycle, bus, motorcycle, pickup truck, train |
| **19** | vehicles 2 | lawn-mower, rocket, streetcar, tank, tractor |

The superclasses and classes are numbered in alphabetical order as shown below:

```
**********CIFAR-100 Superclasses**********
 0 aquatic_mammals                  1 fish
 2 flowers                          3 food_containers
 4 fruit_and_vegetables             5 household_electrical_devices
 6 household_furniture              7 insects
 8 large_carnivores                 9 large_man-made_outdoor_things
10 large_natural_outdoor_scenes    11 large_omnivores_and_herbivores
12 medium_mammals                  13 non-insect_invertebrates
14 people                          15 reptiles
16 small_mammals                   17 trees
18 vehicles_1                      19 vehicles_2

**********CIFAR-100 Classes**********
 0 apple            1 aquarium_fish    2 baby             3 bear             4 beaver
 5 bed              6 bee              7 beetle           8 bicycle          9 bottle
10 bowl            11 boy             12 bridge          13 bus             14 butterfly
15 camel           16 can             17 castle          18 caterpillar     19 cattle
20 chair           21 chimpanzee      22 clock           23 cloud           24 cockroach
25 couch           26 crab            27 crocodile       28 cup             29 dinosaur
30 dolphin         31 elephant        32 flatfish        33 forest          34 fox
35 girl            36 hamster         37 house           38 kangaroo        39 keyboard
40 lamp            41 lawn_mower      42 leopard         43 lion            44 lizard
45 lobster         46 man             47 maple_tree      48 motorcycle      49 mountain
50 mouse           51 mushroom        52 oak_tree        53 orange          54 orchid
55 otter           56 palm_tree       57 pear            58 pickup_truck    59 pine_tree
60 plain           61 plate           62 poppy           63 porcupine       64 possum
65 rabbit          66 raccoon         67 ray             68 road            69 rocket
70 rose            71 sea             72 seal            73 shark           74 shrew
75 skunk           76 skyscraper      77 snail           78 snake           79 spider
80 squirrel        81 streetcar       82 sunflower       83 sweet_pepper    84 table
85 tank            86 telephone       87 television      88 tiger           89 tractor
90 train           91 trout           92 tulip           93 turtle          94 wardrobe
95 whale           96 willow_tree     97 wolf            98 woman           99 worm
```

For my project I did not use all the super classes. To keep the computational time less than CIFAR-100 and more challenging than CIFAR-10, I created a sub dataset, namely CIFAR-15 with reduced super classes.  I have selected the below super classes by considering an example of user searching for “pedestrians” or “vehicles”.

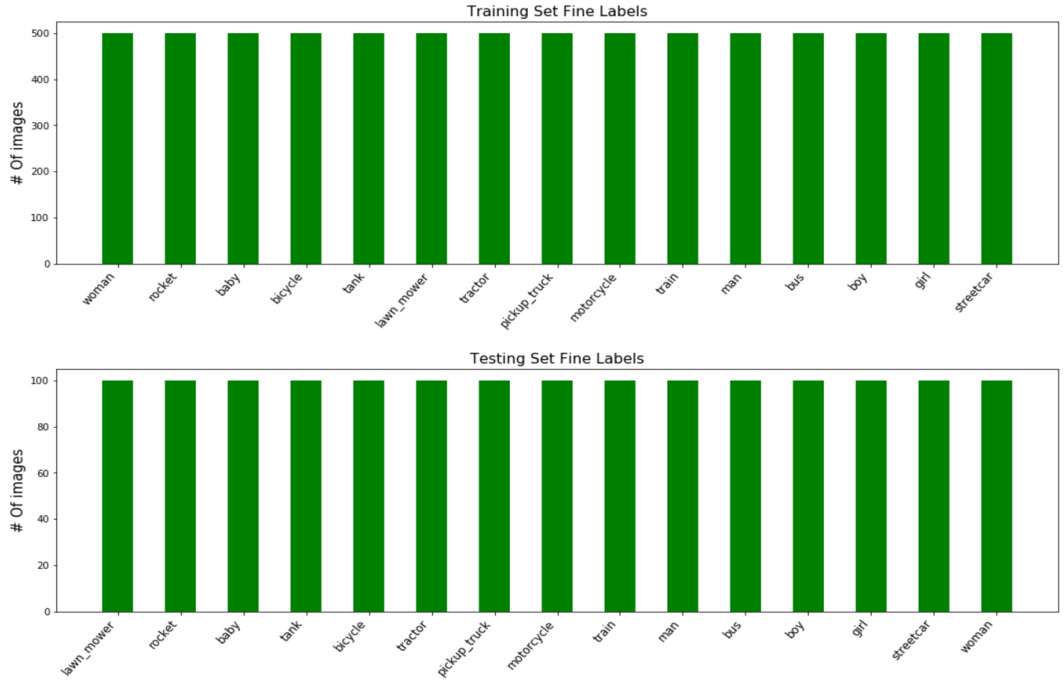
The superclasses and classes in CIFAR-15 are as shown below:

```
**CIFAR-15 Superclasses**
 0 people
 1 vehicles_1
 2 vehicles_2

********************CIFAR-15 Classes********************
 0 baby              1 bicycle           2 boy
 3 bus               4 girl              5 lawn_mower
 6 man               7 motorcycle        8 pickup_truck
 9 rocket           10 streetcar        11 tank
12 tractor          13 train            14 woman
```

## Exploratory Visualization

Below is the distribution of each fine label in the CIFAR-15 dataset:



It can be seen that only the fine labels of the selected super classes are present in the CIFAR-15 dataset. There are 500 images of each class in the training set and 100 images of each class in testing set. The distribution is uniform and hence it eliminates the possibility of model to be biased to one particular class which could have resulted in the model classifying most of the input images as that particular class.

Below are few images from the CIFAR-15 dataset:


It can be seen that some images have some background, like the forest behind the man, or other objects in the image, like toys in the boy image. These might pose some difficulties for the model as the model might lose focus on the main object in the image.

## Metrics

As shown in the Data Exploration section, every image has got a class number, e.g. class "boy" has got class number 2. The class numbers will be one-hot encoded[3] to get a vector of length

[3] http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html

15(maximum number of classes in CIFAR-15), namely y_true. Given an image as input, the model outputs a "prediction" class, namely y_pred, predicting the class the image belongs to.

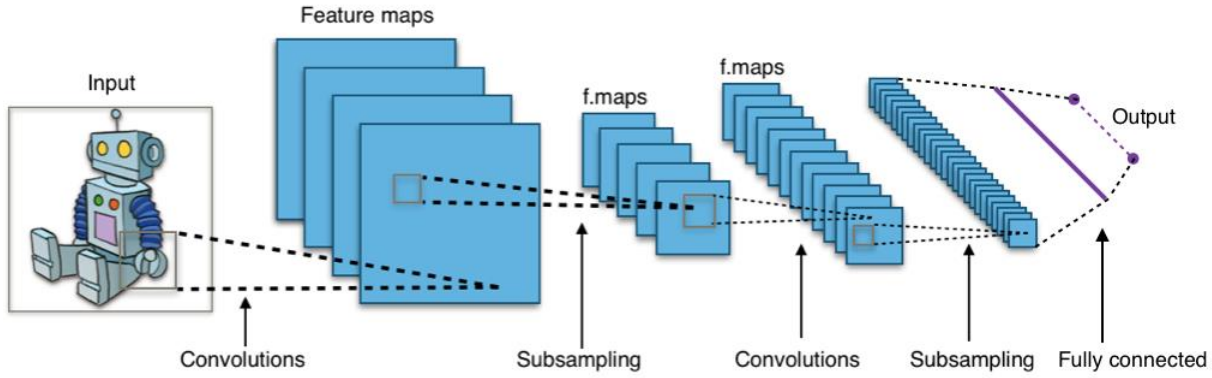
As the dataset is uniformly distributed across all the 15 classes and the problem at hand is that of classification , I used "accuracy" as my evaluation metric. Accuracy is more interpretable and can be loosely stated as "how often does the prediction match the true value". For e.g. if 4 out of 10 predictions match the true value, then the accuracy would be 40%.

Accuracy indicates how good the model is performing but to get an idea of where the model is struggling I used confusion matrix too. It also gives a feel of FP and FN numbers.

## Algorithms and Techniques

Convolutional networks were inspired by biological processes in which the connectivity pattern between neurons is inspired by the organization of the animal visual cortex[4] . CNNs require very little pre-processing compared to other image classification algorithms. This implies that the network learns the filters which in traditional algorithms have to be hand-designed.

Below is an image of a typical CNN:




A typical CNN consists of an input, an output and multiple hidden layers. The hidden layers are either convolutional, pooling/subsampling or fully connected. Convolutional layers apply a convolution operation to the input, passing the result to the next layer. Pooling layers combine the outputs of neuron clusters at one layer into a single neuron in the next layer. Fully connected layers connect every neuron in one layer to every neuron in another layer.

The convolution layer consists of a set of filters. Each filter learns a particular pattern in the image. Combination of these filters extracts a feature of the object. More convolution layers imply more features can be extracted. For an image of a person as an example, one convolution

[4] https://en.wikipedia.org/wiki/Convolutional_neural_network

layer may extract the hands of the person, another convolution layer may extract the eyes, yet another may extract ears and so on.

Pooling layers acts as non-linear down samplers thus reducing the size of input to the next layer. This helps in reducing the number of model parameters and computation time. It is also commonly used to reduce overfitting. Overfitting is the scenario where the model learns the training dataset so well that it performs poorly on unseen data. Pooling can be broadly classified into Max Pooling and Average Pooling. Max Pooling returns the maximum of each sub-region of the input image whereas the Average Pooling returns the average of the sub-region. Max Pooling is the most commonly used pooling function.

Fully connected layers are the regular neural network layer where the neurons of the fully connected layers have connections to all the neurons of the previous layer. The fully connected layers acts as the high-level decision makers. The input to the fully connected layers will be a set of features extracted by the convolution layers and with the help of more fully connected layers, if required, the model can identify the object in the image. For e.g., with the features like eyes, ears, hands and so on the model can identify the object in the image as a human.

Fully connected layers are prone to overfitting as they usually contain a lot of parameters. Dropout is one method to reduce overfitting. In this method, during training stage, individual nodes are dropped out with a certain probability and the training continues on remaining nodes. This avoids training all nodes on all data thereby reducing overfitting and computation time.

The goal of the project was to design a simple CNN model. Simple model implies less memory, less training time and faster execution.

## Benchmark

I used Support Vector Machine(SVM)[5] with Radial Basis Function(RBF)[6] kernel as my benchmark model. SVMs are supervised learning methods used for classification, regression and outliers detection. For obtaining non-linear decision boundaries, SVMs use a kernel function. Usually non-linear kernels, such as RBF, yield better performance for classification.

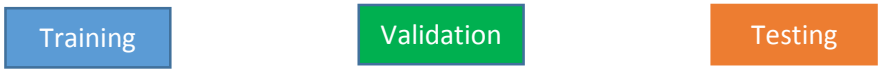
With the default settings of the Sklearn's[7] Support Vector Classification(SVC)[8], the benchmark model took 91 seconds for training and gave a test accuracy of 25.78%.

[5] https://en.wikipedia.org/wiki/Support_vector_machine
[6] https://en.wikipedia.org/wiki/Radial_basis_function_kernel
[7] http://scikit-learn.org/stable/
[8] http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

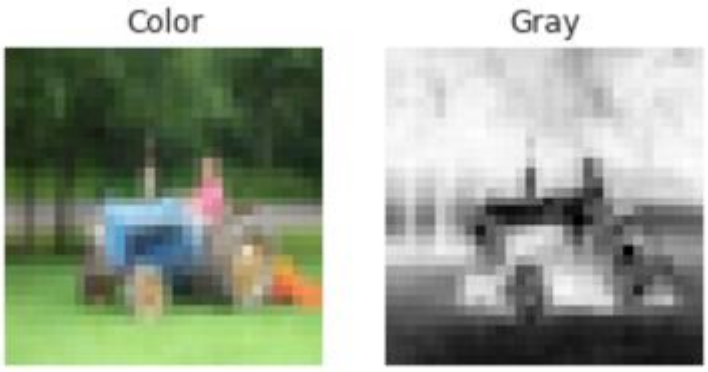
# III. Methodology

## Data Preprocessing

The CIFAR-15 dataset contains training and testing data. A part of training data is set aside as validation data. The remaining training set will be used to train the model, i.e. modify its weights, and validation set will be used to check how the model is doing. As the validation set is not used to modify the model's weights, it also gives a good indication if the model is overfitting to the training set. The testing set will be used to find the final accuracy of the model. The testing set acts like real world data as the model never saw the testing set.




As part of preprocessing, I converted each color image to grayscale. As the goal of the project is to classify objects, the shape of the object plays an important role than its color. Converting to grayscale also reduces the input size helping the model to train faster.




Next I normalized the images. Images of same class objects taken in different lighting condition will lead to a huge difference in pixel values but I want my model to classify both these images as same class. Normalizing the images will make sure that all the images have pixel values in similar range. I normalized the images to be in the range 0 to 1.

## Implementation

I used Keras[9], the python library for deep learning, for implementing my CNN model. I used the Amazon AWS GPU Instance[10] to train and test my model.

[9] https://keras.io/
[10] https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/accelerated-computing-instances.html

Keras provides the below functions to implement different layer:

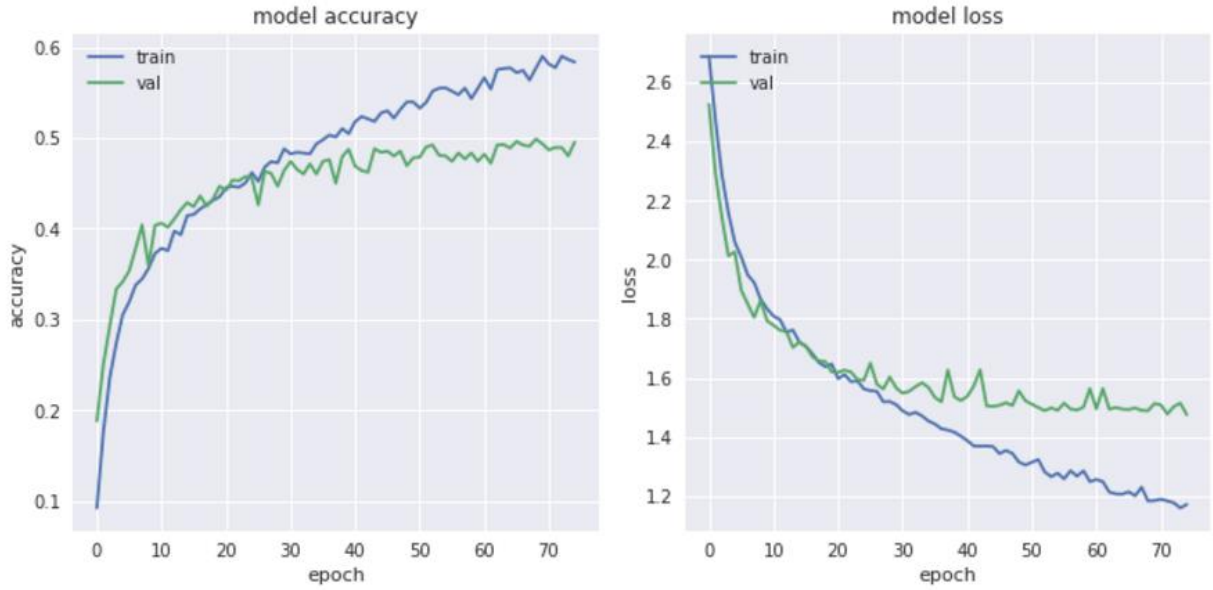
| Keras Function | Purpose |
|---|---|
| **Conv2D** | Applies 2D convolution on images |
| **MaxPooling2D** | Applies max pooling operation on images |
| **Flatten** | Flattens the higher dimensional input to single dimension |
| **Dense** | Creates a regular Fully-connected NN layer |
| **Dropout** | Applies dropout on the input |

Below is my final model after a lot of trial and error to improve performance:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 16)        80
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 32)        2080
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 32)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 8, 8, 64)          8256
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1024)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               262400
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 15)                3855
=================================================================
Total params: 276,671
Trainable params: 276,671
Non-trainable params: 0
_________________________________________________________________
```

The drop rate of the first dropout layer is 0.5 and of second is 0.7.

Below is the performance of the model across 75 iterations:




```
Time to train CNN is 231.05269312858582 seconds.
```

I used the Keras built-in Adam[11] optimizer to update the network weights. As this project deals with multi-class classification I used the Keras built-in Categorical Cross Entropy[12] loss function to compute the cross-entropy between prediction and target. I used accuracy as my metric to judge the performance of my model.

I used the confusion matrix to show each class in the evaluation data and the number or percentage of correct predictions and incorrect predictions.

## Refinement

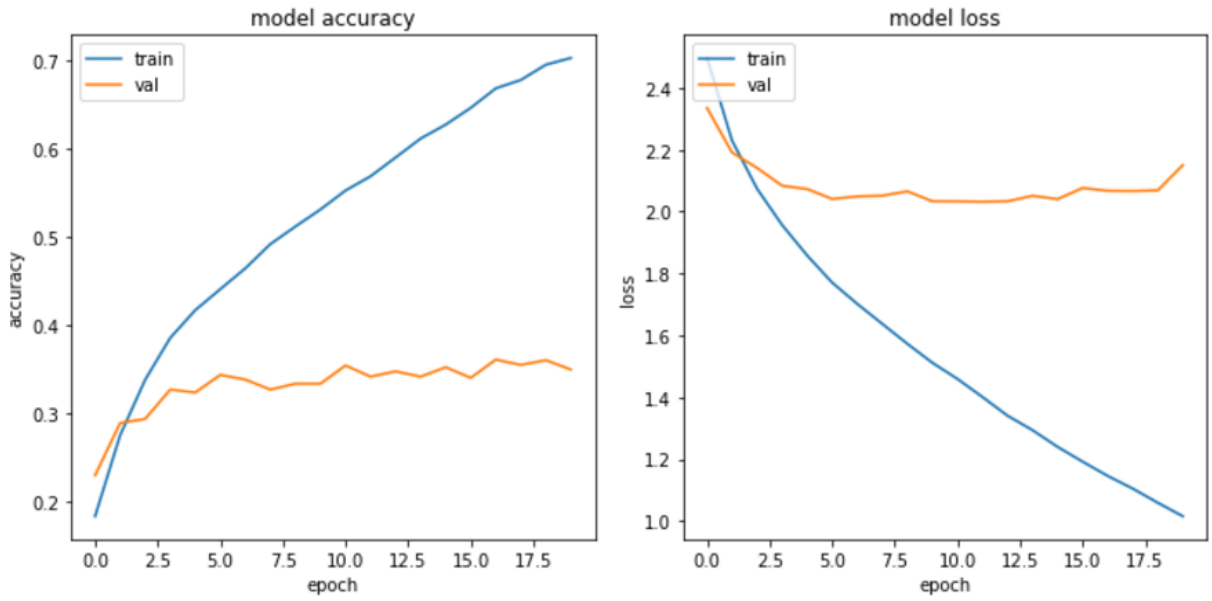
I reached my final model by building upon my initial model, which had one convolution and one fully connected layer, and checking the model's performance with every modification. I experimented by adding different layers, changing the number of filters in the convolution layer, changing the number of neurons in the dense layer, tuning the drop rate of the dropout layers and changing the number of iteration or epochs to check how the model performed.

[11] https://en.wikipedia.org/wiki/Stochastic_gradient_descent#Adam
[12] http://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html#cross-entropy

My initial model was as below:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 8)         40
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_1 (Dense)              (None, 15)                122895
=================================================================
Total params: 122,935
Trainable params: 122,935
Non-trainable params: 0
_________________________________________________________________
```

This model gave a test accuracy of around 35% which was already better than the test accuracy of the benchmark model. Below is the performance of the model across 20 iterations:




```
Time to train CNN is 25.480663776397705 seconds.
```

Though the accuracy of the model was better than that of the benchmark model, the graphs above told a different story. The model was overfitting.

It was observed that by adding more convolution layers the model performed better on training data but had the effect of overfitting. Introducing max pooling layers successfully reduced overfitting but at the cost of learning time. Adding fully connected layer did provide small

improvement on the learning curve but also led to overfitting and more computation time as the number of parameters increased drastically due to fully connected layer. So a trade-off had to be made on number of fully connected layers and computation time. Adding dropout layer had similar effect of max pooling where the overfitting was reduced but with slow learning curve. Playing with the dropout rate showed that with lower dropout rate (less than 0.5), there was not much improvement in terms of accuracy and computation time. Higher dropout rate (more than 0.5) reduced overfitting and also computation time but had negative effect on learning curve. Increasing the number of neurons in the fully connected layer was beneficial but at the cost of computation time and hence a trade-off was required. Increasing the number of iterations, after eliminating any overfitting, was helpful in better performance of the model but at the cost of training time. Augmenting data was too costly in terms of training time. Augmenting with random rotation alone costed 2 hours of training time so I decided against augmentation. I finally decided to settle on the model which had a close to double the accuracy of the benchmark model.

# IV. Results

## Model Evaluation and Validation

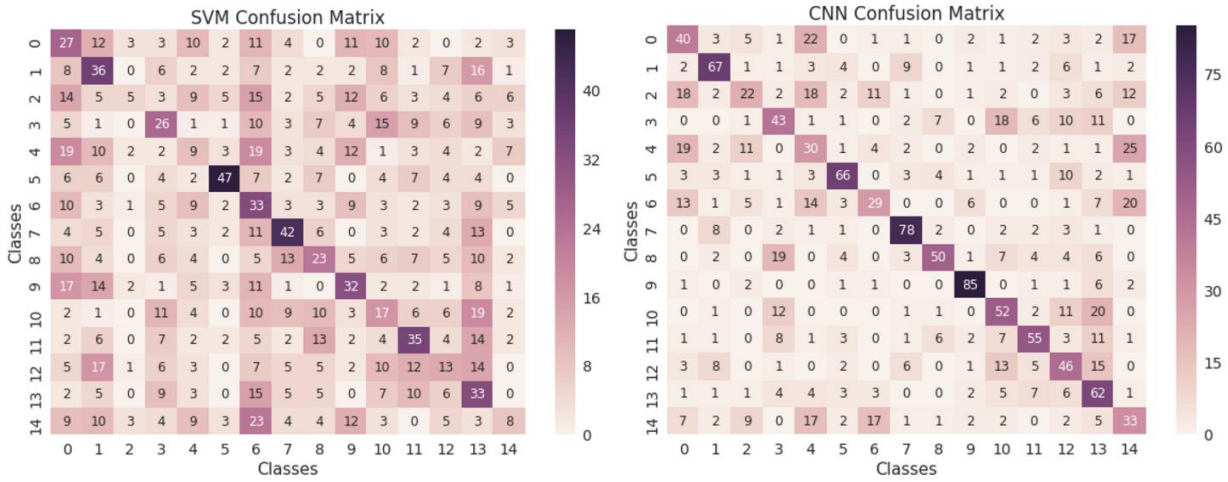
The final model was decided after a lot of experimentation on using different neural network layers and tuning different parameters as mentioned in the Refinement section above. The model was validated on validation set to check for overfitting on the training set. A trade off had to be made between the accuracy and computation time to decide the final model. Later the model was tested on testing set which was never seen by the model before and it performed exceptionally well when comparted to the benchmark model. A comparative result is provided in the Justification section below.

## Justification

Comparing the accuracies of the benchmark model and the final CNN model we see that the CNN model performs far better than the benchmark model, actually twice better than the benchmark model.

| Model | Accuracy(%) |
|---|---|
| SVM | 25.78 |
| CNN | 50.53 |

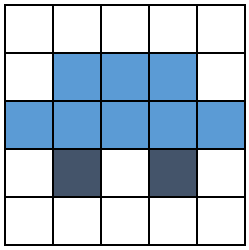
The confusion matrix also shows that the CNN model is doing really good compared to the benchmark model as shown by the darker shade on the diagonal axis. The higher number on the diagonal axis say that the CNN model is able to classify more images correctly. This also implies that the overall FP and FN rate of CNN model is less.




# V. Conclusion

## Free-Form Visualization

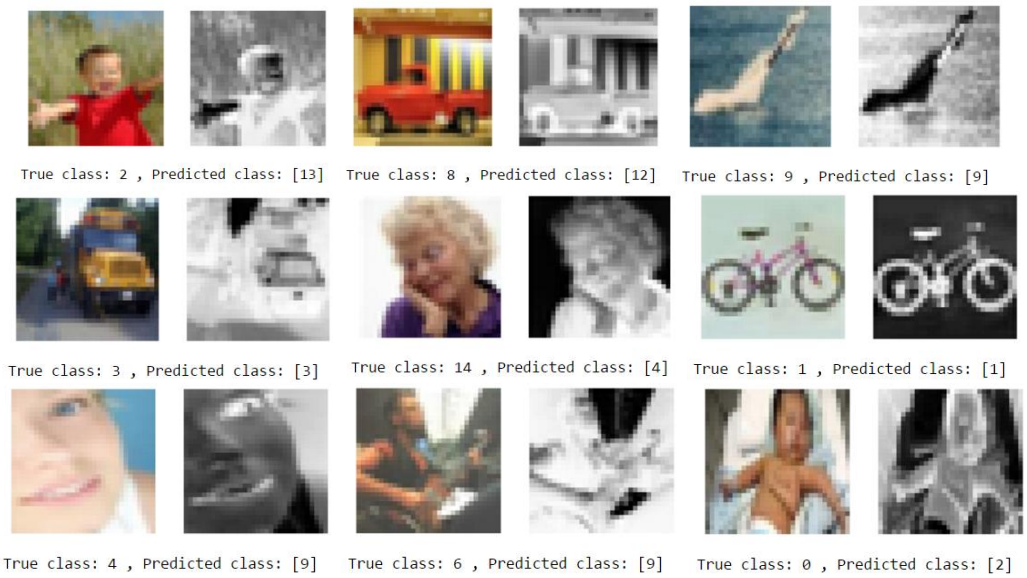
The most important quality of the CNN model is that it is able to accept the images directly whereas the benchmark model needs the input as 1D array thereby losing spatial information. Consider the below example of a 5x5 image of a car:



The same image when converted to 1D looks like below:

Any resemblance to a car is totally lost as the pixels are moved spatially. This makes it difficult for the SVM to learn that the image is of a car. The CNN on the other hand will have all the spatial information intact to start with thereby helping it to learn easily.

Below are few predictions of the CNN model:




The above predictions are in agreement with the confusion matrix. Mainly the classes 2, 4, 6 and 14 have lighter shades on the diagonal axis of the confusion matrix indicating that the model is having difficult time classifying these classes correctly.

## Reflection

The problem at hand was to design a CNN for classifying object for Image Search application. An SVM was used as benchmark to check the CNN's performance. The CNN did exceptionally well when compared to the SVM. The most interesting aspect of the project was using different layers of the CNN and knowing their contribution to the model's performance. Finding the right drop rate for the dropout layer turned out to be a difficult task. The CNN model can be used for object classification for images similar to the CIFAR-100 dataset.

## Improvement

The main limitation I faced was lack of computation power. As the number of parameters in the model increases the more time it takes to train the model. So with higher computation power, the model can be improved by adding more dense layers with more neurons. This will help the model in mapping a set of features to an object easily. Dataset augmentation is another important factor to be considered for improving the model's performance. With more data the model should be able to classify correctly the images which the current model is struggling with,

as seen by the confusion matrix. Overall, with smart choice of dense layers and data augmentation, the improved model should be able to perform even better than the current model.

# VI. References

[1] Sandeep Kumar, Zeeshan Khan, Anurag Jain, A Review of Content Based Image Classification using Machine Learning Approach, International Journal of Advanced Computer Research Volume-2 Number-3 Issue-5 September-2012

[2] Er. Navjot Kaur and Er. Yadwinder Kaur, Object classification Techniques using Machine Learning Model, International Journal of Computer Trends and Technology (IJCTT) – Volume 18 Number 4 – Dec 2014

[3] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, Martin Riedmiller, Striving For Simplicity: The All Convolutional Net, Workshop contribution at ICLR 2015