Machine Learning Engineer Nanodegree

Capstone Project

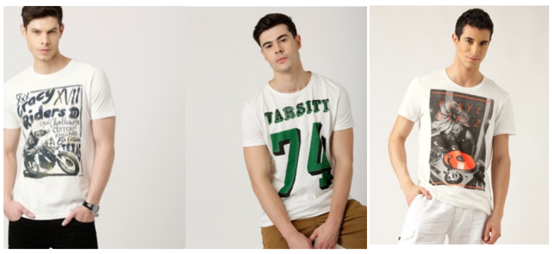
Prateek Gupta
July 7th, 2018

I. Definition

Project Overview

In 1995's Clueless, you may recall Cher Horowitz using cutting-edge software to select her plaid ensemble. Cher's machine could identify chic head-to-toe looks, adding a small dose of sci-fi to the rom-com classic. Twenty-two years later, the 90's fiction movie is closer than ever to reality. Technology is already reshaping the fashion industry, from enabling online purchases through mobile devices to powering live streaming of runway shows. However, this innovation is just the tip of the iceberg: the fashion industry is going to change fast and change forever. One of the biggest drivers of technological innovation will be machine learning (ML) and eventually artificial intelligence (AI). Algorithms will foresee what people want to wear, before they're even aware themselves and will recommend outfits and looks based on personal, local and global trends. Online fashion will be transformed by a tool that understands taste. Because if you understand taste, you can delight people with relevant content and a meaningful experience. Outfits are the asset that would allow taste to be understood, to learn what people wear or have in their closet, and what style each of us like.

Customers can purchase wide variety of fashion products like T-shirts from an e-commerce platform. One of the major fashion attributes of a T-shirt is the type of graphic present on it. Graphic type plays a major role in identifying T-shirt categories and determining its saleability. Some example graphic types are:



Biker Varsity Music

The challenge is to identify the graphic type of a T-shirt. Take in an image as an input and give graphic type of the T-shirt as the output. The model can be used during cataloguing new T-shirts on any e-commerce platform. With graphic type present as an attribute, customers will be able to search and filter for different type of T-shirts.

Problem Statement

The dataset was *initially* provided as the files **train.csv** and **test.csv**, with the information necessary to make a prediction. The dataset was gathered and labelled by a team internal to my organisation, by identifying the graphic type for each T-shirt captured from various angles.

The format of each of the file is as follows:
<brand_name>, <article_type>, <gender>, <color>, <image_path>, <graphic_type>
Since, we're interested in only classifying the T-shirt images against their graphic type, I developed a small Java application to download the images and arrange them against their respected graphic type. The dataset features **24** different classes of graphic types for T-shirts. Training set (train.csv) included **30000 rows** (labelled images) and the testing set (test.csv) included **70000 rows**.

To train such a huge dataset, machines with higher computational power are needed, preferably with GPU, which I didn't have access to at that point. Hence, I had to cut the training set to **8266** images (**train**=6676; **valid**=1580) and the testing set to **3560** rows.

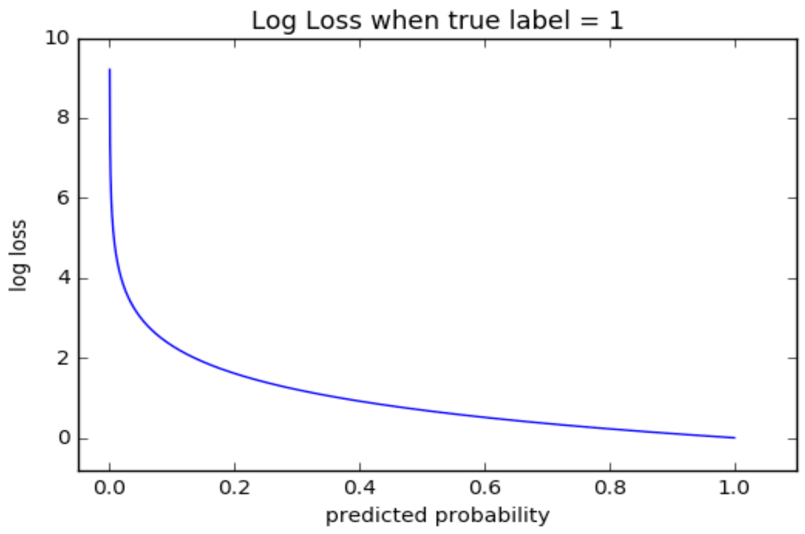
As deep learning techniques have been very effective in image classification over the years, such as, a convolutional neural network (CNN), which is very effective at finding patterns within images by using filters to find specific pixel groupings that are important. In this project, transfer learning along with data augmentation will be used to train a convolutional neural network to classify images of T-shirts to their respective classes. Transfer learning refers to the process of using the weights from pre-trained networks on large dataset. Fortunately, many such networks such as RESNET, Inception-V3, VGG-16 (created by Oxford's Visual Geometry Group) pre-trained on *ImageNet challenge* are available for use publicly.

Metrics

The metric used for this project is multi-class logarithmic loss (also known as **categorical cross entropy**). Here each image has been labelled with one true class and for each image a set of predicted probabilities should be submitted. **n** is the number of images in the test set, **m** is the number of image class labels, $\mathbf{y_{ij}}$ is 1 if observation **i** belongs to class **j** and **0** otherwise, and $\mathbf{p_{ij}}$ is the predicted probability that observation **i** belongs to class **j**. A perfect classifier will have the log-loss of 0.

$$\mathcal{L}(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(p_i) + (1-y_i)\log(1-p_i)\right] = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij}\log(p_{ij})$$

Multiclass log-loss punishes the classifiers which are confident about an incorrect prediction. If the class label is 1 (the instance is from that class) and the predicted probability is near to 1 (classifier predictions are correct), then the loss is really low, so this instance contributes a small amount of loss to the total loss and if this occurs for every single instance (the classifiers is accurate) then the total loss will also approach 0. On the other hand, if the class label is 1 (the instance is from that class) and the predicted probability is close to 0 (the classifier is confident in its mistake), as *log(0)* is undefined it approaches infinity, so theoretically the loss can approach infinity.



II. Analysis

Data Exploration

Each image correlated with a link is in the *jpeg* format. The resolution of each image is **1080 x 1440**, with the DPI of **72 pixels/inch**. The dataset features **24** different classes of graphic types for T-shirts, under different lighting conditions, from different angles and different activity done by the models wearing them. However, it's closer to the real life, so any system for graphic type classification of a T-shirt must be able to handle this sort of data. Images do not contain any border. Each image has only one type of graphic category, except that there are sometimes the t-shirts hang on a hanger. Training set includes **8266** labelled images (**train**=6676; **valid**=1580) and the testing set includes **3560** rows. Images are guaranteed to be of fixed dimensions. The dataset was labelled by identifying the graphic type for each T-shirt captured from various angles.

- **Features**
  The features I'm going to use are represented by each individual pixel in the images. The CNN classifier works with coloured image, so there is a third dimension that has to be added to each pixel. This third dimension is an array with 3 elements, each one representing the Red, Blue and Green channel of the pixel.
- **Labels**
  Each image in the dataset is associated with a label, which is an integer ranging from 0 to 23. This label is actually an index of the class description in a list that contains the 24 classes.
- **Label Distribution**
  The dataset is imperfectly distributed amongst the 24 labels. The no. of images ranges from *tens* for a few labels to *a few thousands* for others.

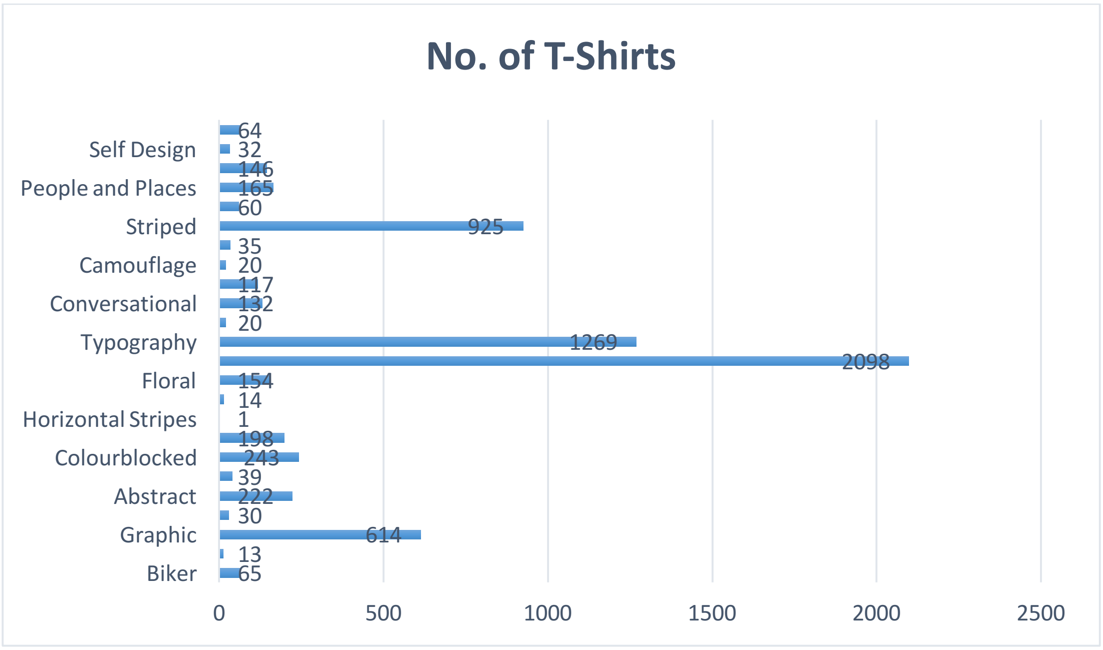
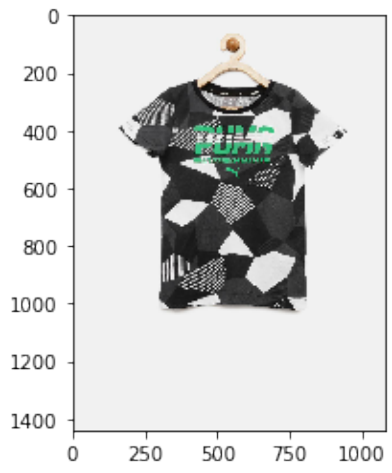
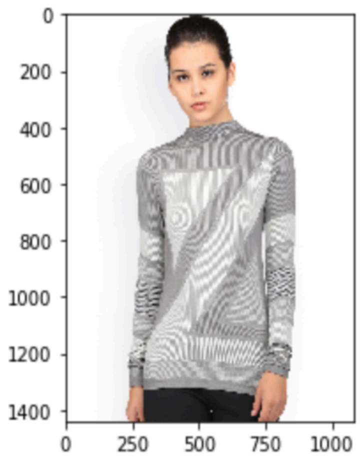


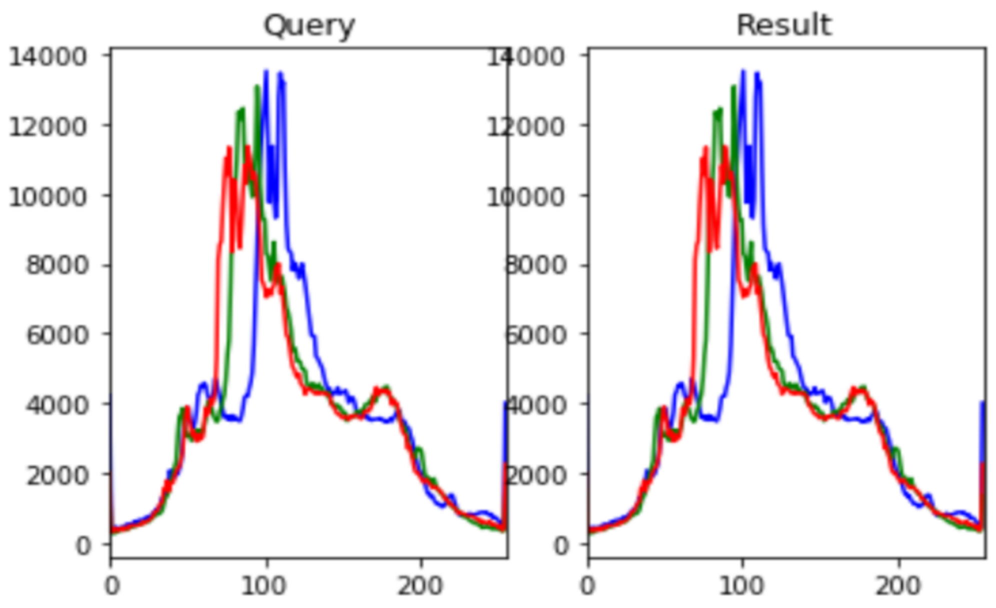
Figure 1: Label distribution

Exploratory Visualisation

Histograms represent the colour distribution of an image by plotting the frequencies of each pixel values in the 3 colour channels. In image classification histograms can be used as a feature vector with the assumption that similar images will have similar colour distribution. Here, we *can* calculate the histogram for a image in the training set and find the result for the most similar image from the histograms with the Euclidean distance metric. Results for a randomly chosen sample image is given below:

Clearly the images are similar in the labels, but they don't look similar. However, their histograms are quite similar. Note that a benchmark model with **k-nearest neighbours** *can* also be trained with the colour histograms as features. However, histograms completely ignore the shape, texture and the spatial information in the images and very sensitive to noise, so they can't be used to train an advanced model.



Algorithms and Techniques

The main goal of this project is to build a Convolutional Neural Network (CNN) classifier to predict image labels.

Layers:

- **Convolution**: Convolutional layers convolve around the image to detect edges, lines, blobs of colours and other visual elements. Convolutional layers hyper-parameters are the number of filters, filter size, stride, padding and activation functions for introducing nonlinearity.
- **MaxPooling**: Pooling layers reduces the dimensionality of the images by removing some of the pixels from the image. MaxPooling replaces a n x n area of an image with the maximum pixel value from that area to down sample the image.
- **Dropout**: Dropout is a simple and effective technique to prevent the neural network from over-fitting during the training. Dropout is implemented by only keeping a neuron active with some probability p and setting it to 0 otherwise. This forces the network to not learn redundant information.
- **Flatten**: Flattens the output of the convolution layers to feed into the Dense layers.
- **Dense**: Dense layers are the traditional Fully Connected networks that maps the scores of the convolutional layers into the correct labels with an activation function.

Activation Functions:

Activation layers apply a nonlinear operation to the output of the other layers such as convolutional layers or dense layers.

- **ReLu Activation**: ReLu or Rectified Linear Unit computes the function $f(x)=max(0,x)$ to threshold the activation at 0.

- **Softmax Activation**: Softmax function is applied to the output layer to convert the scores into probabilities that sum to 1.

Optimizers:

- **RMSprop**: running average of both the gradients and their magnitude is used. In my experiments RMSprop also shows general high accuracy. I've used RMSprop in all the experiments because I felt having similar optimizer would be a better baseline for comparing the experiments.

The convolution operation corresponds to a filter operation in the image processing. It moves the window of the filter at regular intervals and applies it to the input data.

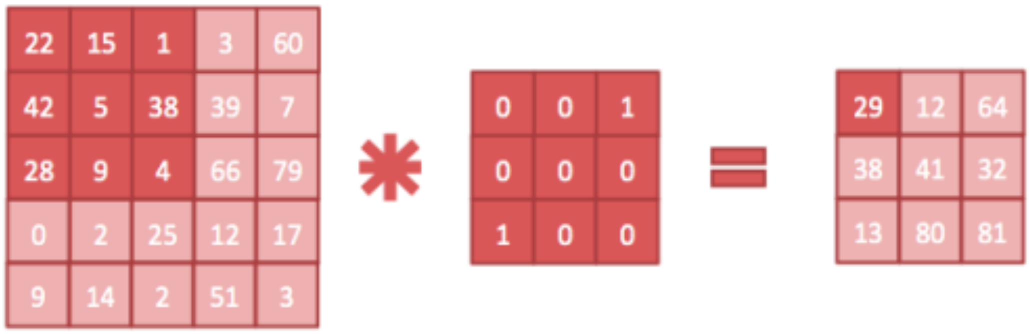


Figure 2: Convolution Operation

As shown in the figure above, the convolution operation applies a filter to the input data. In this example, the input data has dimensions in the vertical and horizontal directions. In this example, the input is (5,5), the filter is (3,3), and the output is (3,3). Filters may also be referred to as kernels, depending on the literature.

The convolution operation moves the window of the filter at regular intervals and applies it to the input data. The window here refers to the dark 3x3. As shown in this figure, multiply the input elements by the corresponding elements in the filter and obtain the sum. This process is performed everywhere and stored in the corresponding place in the output, and the output of
the convolution operation is completed.

I have multiple of CNV layers in the CNN image recognition and input to these CNV layers are called Input Feature map and output of the CNV layers are called Output Feature map. At the very first CNV layer, input image with its each, component becomes input feature maps with 3 2D data. Figure 3 illustrates high-level input output feature map structure with LxL CNV filters. K convolutional filter output is combined to generate one output feature map.

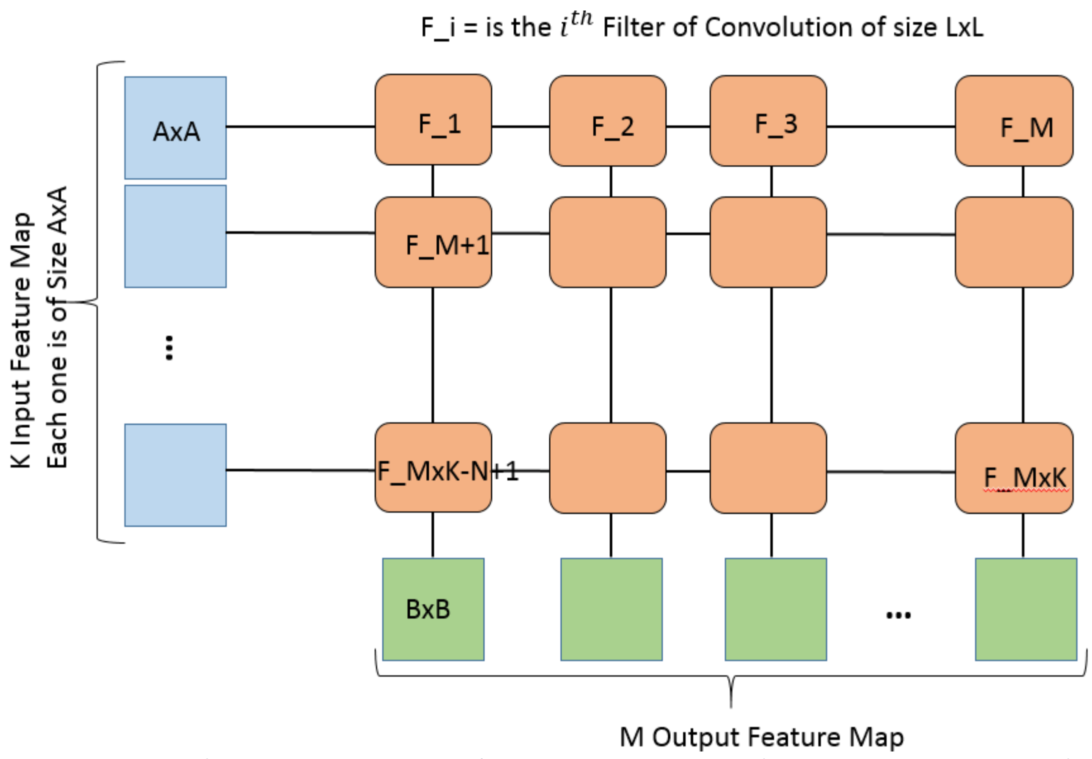
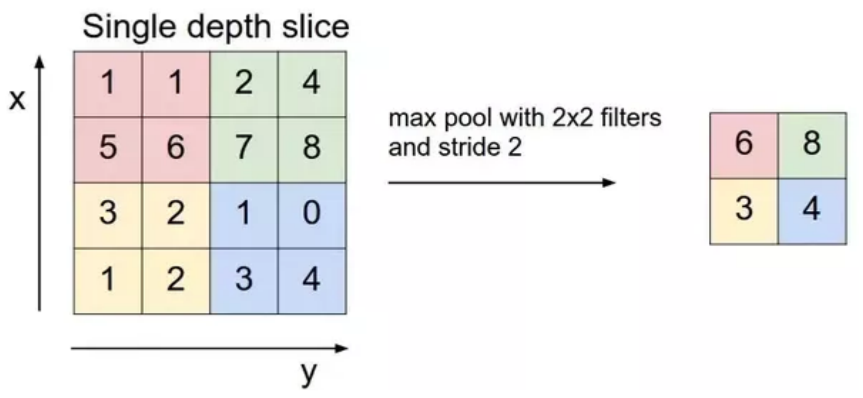


Figure 3: Input and Output Feature Map with K input Feature Map and M Output Feature Map and LxL Convolutional Filter

After the Convolution layer is applied, I can execute Pooling. This layer will reduce the image a little bit, by creating smaller, aggregated sections of the stacks. Pooling will walk through the zones of the image and pick the highest scored pixel on that zone. This process helps the detection process to assimilate variances on the selected feature, like small differences in positioning and tilting. Window size that is used during subsampling is also 2D such as 2x2 or 3x3. If the input data size is (W,H) then the size after pooling will be (W/2,H/2) if ½ subsampling is used. For reference, it is common to set the window size and stride of the pooling to the same value.



The choice of which layers to use, and their order, how many neurons to add to each layer are part of the art of building a CNN. More skilled engineers can even build custom layers, tailored specifically to the problem they're dealing. The output of each layer becomes the input for the next. At first, each neuron from a given layer is connect to all the neurons from the next layer. This creates a complex network of connections, where each connecting line has a specific weight that controls the significance of that particular output to the next layer. One of the main jobs of the CNN algorithm, implemented by Keras, is to define those weights.

The training process applies a somewhat simple technique to find the optimal weights for each neuron, called **backpropagation**. The algorithm will work with already labelled images (**train** set) and calculate the error rates for each prediction. Based on the error rate, new weights that reduce that rate will be define. After the training is complete, some neurons might have been given weights equals to 0. The output connection for those neurons are ignored, resulting on a simpler network.

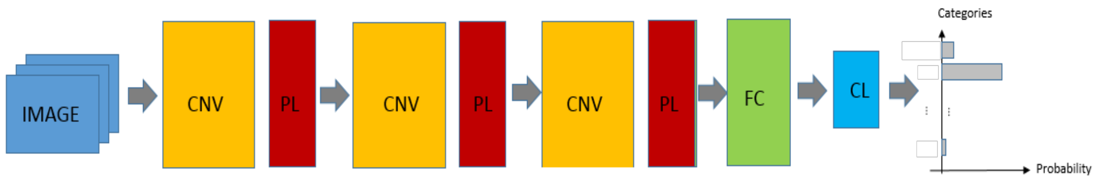
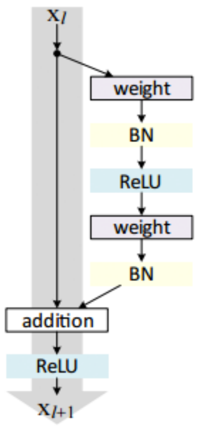


Figure 3: CNN architecture with 3CNV and 1 Fully Connected layers

Transfer Learning:

It uses the weight values learned from the huge dataset provided by ImageNet for the dataset of this project. Weights copied from the ImageNet are copied to another neural network, and fine tuning is performed in that state. In this project, ResNet50 is prepared, the previously learned weight is set as the initial value, and fine tuning is performed on the new dataset.

ResNet50 introduces is residual learning. In residual learning, instead of trying to learn some features, we try to learn some residual. Residual can be simply understood as subtraction of feature learned from input of that layer. ResNet50 does this using shortcut connections (directly connecting input of nth layer to some (n + x)th layer. It has proved that training this form of networks is easier than training simple deep convolutional neural networks and also the problem of degrading accuracy is resolved when it is judged that training data is insufficient.

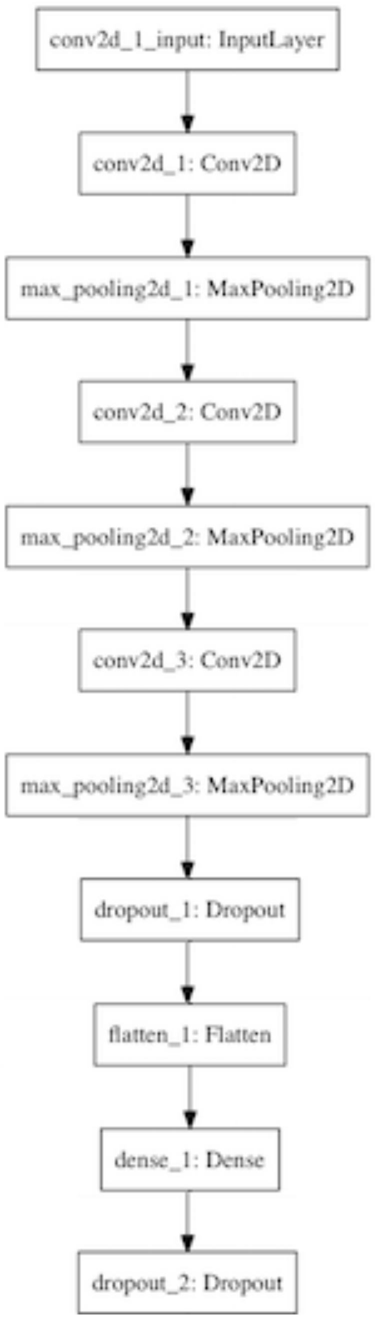


The residual module in ResNet

Benchmark

Random choice: We predict equal probability for a T-shirt to belong to any class of the 24 classes for the naive benchmark.

CNN: A well-designed convolutional neural network should be able to beat the random choice baseline model easily. This model consists of three convolution layers. The max pooling is performed for each convolution layer. Of the last two output layers, the first layer uses the ReLu activation function and the last layer uses the Softmax activation function. The figure below shows the CNN architecture.



The accuracy for the benchmark model is 87.03%.

III. Methodology

Data Preprocessing

Before feeding the dataset images into our Sequential Model, there is a minor preparation. I’ve normalized the data, dividing the RGB values by the maximum value, 255. This results on decimal numbers from 0 to 1.
And the benefits are:

- **Treat all images in the same manner**: Some images are high pixel range; some are low pixel range. The images are all sharing the same model, weights and learning rate. The high range image tends to create stronger loss while low range create weak loss, the sum of them will all contribute the back propagation update. But for visual understanding, you care about the contour more than how strong is the contrast as long as the contour is reserved. Scaling every images to the same range [0,1] will make images contributes more evenly to the total loss. Without scaling, the high pixel range images will have large amount of votes to determine how to update weights. For example, black/white cat image could be higher pixel range than pure black cat image, but it just doesn't mean black/white cat image is more important for training.
- **Using typical learning rate**: When we reference learning rate from other's work, we can directly reference to their learning rate if both works do the scaling preprocessing over images data set. Otherwise, higher pixel range image results higher loss and should use smaller learning rate, lower pixel range image will need larger learning rate.

Implementation

The list for the required libraries is below.

- Jupyter Notebook
- Python 3.6
- Keras
- Tensorflow
- numpy
- pandas
- matplotlib
- plotly

I’ve used Keras’ Sequential Model. This model allows us to use many processing layers that run sequentially and improve the accuracy of the classifier on each iteration, or epoch. In my case I’ve implemented a pattern using the following steps:
1) Convolution2D and MaxPooling2D (repeated 3 times), increasing the number of feature maps from 16 to 32 and, finally, to 64.
2) Dropout layer to prevent over-fitting.
3) Flatten layer to *flatten* the output of the convolution layers to feed into the Dense layer.

4) Dense layer to map the scores of the convolutional layers into the correct labels with ReLu activation function.
5) Dropout layer to prevent over-fitting.
6) Dense layer to map the scores of the convolutional layers into the correct labels with Softmax activation function.
7) In order to find the parameters that make the value of the loss function as low as possible, it is important to find the optimal value of the parameter. Solving this problem is called optimization. I use RMSprop optimizer (divide the gradient by a running average of its recent magnitude) for this purpose.

The execution of layers is applied **5** times.

Training Time

The training took <u>too long (40 min per epoch)</u> to complete. I couldn't run the algorithm on a more powerful machine due to *personal* constraints.
In the end, I had to cut down the dataset I initially started with (as mentioned in the proposal submitted earlier against the project). The result, however, was a <u>classifier with 18.90% accuracy</u>, which is still not up to production standards, but suitable for this project.

Refinement

Since the data set is small (only 6676 training images) it's definitely plausible our model is memorizing the patterns. To overcome this problem, data augmentation can be used. Data Augmentation alters our training batches by applying random rotations, cropping, flipping, shifting, shearing etc. In the specific dataset, random cropping does not make sense because there are images with t-shirt on a hanger, which is small compared to the whole photo and cropping the photos might create a situation where the model starts inferring most of the photo as 'no graphic' class because the t-shirt was cropped away during data augmentation. Vertical flipping does make sense here. I could have added random rotation because it's possible the camera is going to move from one corner to another to cover a broad area. I could have also added horizontal flipping and random shifting up and down and side by side because all these scenarios are likely. *Unfortunately,* the model with data augmentation is computationally expensive, and I wasn't able to proceed with the modifications. ☹

As a final model, this project was carried out using the transfer learning method using ResNet50 model. It has proved that training this form of networks is easier than training simple deep convolutional neural networks and also the problem of degrading accuracy is resolved. It also uses **k-fold** cross validation to prevent over-summing and solve the problem of insufficient data volume to increase the statistical reliability of the classifier performance measurement. The following arguments are set for the ResNet50 model.

```
model = applications.resnet50.ResNet50(include_top=False, weights='imagenet')
generator = datagen.flow_from_directory(train_data_dir, target_size=(224, 224),
batch_size=20, class_mode='categorical', shuffle=False)
```

- weights: 'imagenet' (pre-training on ImageNet).
- include_top: whether to include the 3 fully-connected layers at the top of the network.

StratifiedKFold cross validation object is a k-fold variant that returns a stratified fold. Folds are created by maintaining a sample rate for each class.

```
folds = list(StratifiedKFold(n_splits=K, shuffle=True).split(train,valid))
```

- n_splits : number of folds. Must be at least 2.
- shuffle : whether to shuffle each stratification of the data before splitting into batches.

Complications that occurred during the coding process

1. Time consumption

   When using my MacBook Pro, the learning time was spent about two hours. Also, CUDA could not be installed because the MacBook Pro GPU is not NVIDIA. To solve this waste of time, I could not use AWS EC2 instance due to personal reasons.

2. ResNet50

   Finally, I have encountered difficulties in applying the ResNet50 model to transfer learning. I referred to Keras Documentation and the previous project, Dog Breed Classifier.

Results

1. Benchmark Model (Random): 4.35%
2. Benchmark Model (CNN Model): 18.90%
3. Transfer Learning with ResNet50 Model: 84.75%

IV. Results

Model Evaluation and Validation

I think that using the ResNet50 model for transfer learning along with k-fold cross validation (to improve the statistical reliability of the classifier performance measurement by resampling model) to increase the test set accuracy to 84.75% is a reasonable project result. In transfer learning, we take a pre-trained model (network + weights), and then remove the Fully Connected network, and construct our own in place of it. Doing so, it will remove the pre-trained weights at those layers. Now the original network before the Fully Connected network is frozen so that, when the training is started on the new data set, only the newly added Fully Connected network is trained. Here the input to the Fully Connected network is called the **bottleneck features**. They represent the activation map from the last convolutional layer in the network.

Hence with frozen weights in the main network, we will get to use the pre-trained weights to get important feature activations as bottleneck features into our newly added Fully Connected network, and now this new Fully Connected network (trained on our data) gives us the required inference as per training.

Choice of k in k-Fold Cross Validation

In order to select the most suitable number of k for the model, we derive the result by changing the value of k as follows.

1. k = 3
   The mean of the accuracy at each fold is: 72.14% (0.7214346564676)
   The variance value is:0.000186279905974

2. k = 5
   The mean of the accuracy at each fold is: 73.73% (0.7372894656644)
   The variance value is: 0.0240667066218

3. k = 7
   The mean of the accuracy at each fold is: 82.20% (0.8220006539197)
   The variance value is: 0.00040460088166

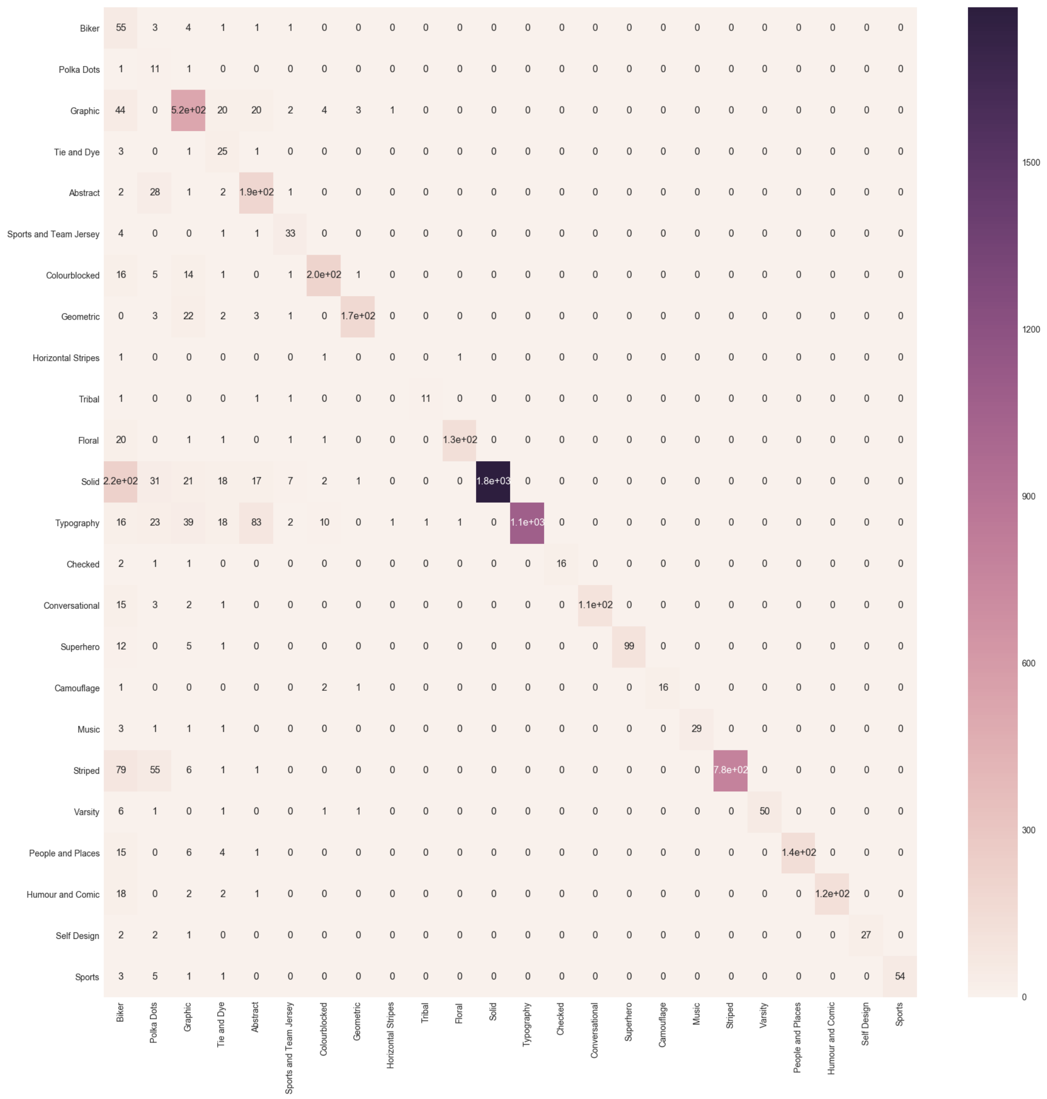
4. k = 10
   The mean of the accuracy at each fold is: 84.75% (0.8474586822329)
   The variance value is: 0.000481258375864

The highest test accuracy among the total of 4 experiments is when k is 10. The smaller the Log Loss Validation value is, the better the result. When k is 7 and 10, the latter has slightly better results. Therefore, k-fold CV when k is 10 is the most robust model. However, when I check each variation value, the variation is smallest at k = 3, and 7 and 10 are not much different from each other. This result is contrary to the common saying, "the smaller the k, the greater the variance". Nevertheless, since the lowest value of test Log Loss is 10, I think that the model with k = 10 is the most robust.

To validate the model, I generated predictions for the validation data which had an accuracy score of 84.75%. In the validation data out of 6676 images, 5645 images are classified accurately and 1031 images are incorrect.

The confusion matrix (non-normalized) plot of the predictions on the validation data is given below (figure 4).

As seen from the confusion matrix, this model is really good at predicting Solid and Typography classes, presumably because the training data
provided itself has more Solid and Typography photos than other classes. However, its possible that the provided dataset is imbalanced because it's the accurate reflection of the demand of graphic types of T-shirts in online fashion retail.



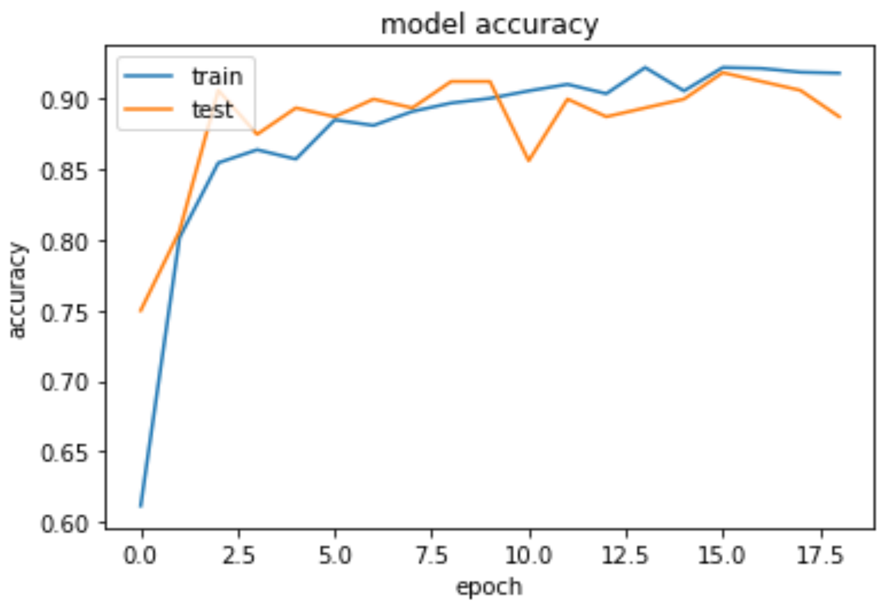
	Biker	Polka Dots	Graphic	Tie and Dye	Abstract	Sports and Team Jersey	Colourblocked	Geometric	Horizontal Stripes	Tribal	Floral	Solid	Typography	Checked	Conversational	Superhero	Camouflage	Music	Striped	Varsity	People and Places	Humour and Comic	Self Design	Sports
Biker	55	3	4	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Polka Dots	1	11	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Graphic	44	0	5.2e+02	20	20	2	4	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Tie and Dye	3	0	1	25	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Abstract	2	28	1	2	1.9e+02	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Sports and Team Jersey	4	0	0	1	1	33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Colourblocked	16	5	14	1	0	1	2.0e+02	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Geometric	0	3	22	2	3	1	0	1.7e+02	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Horizontal Stripes	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Tribal	1	0	0	0	1	1	0	0	0	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Floral	20	0	1	1	0	1	1	0	0	0	1.3e+02	0	0	0	0	0	0	0	0	0	0	0	0	0
Solid	2.2e+02	31	21	18	17	7	2	1	0	0	0	1.8e+03	0	0	0	0	0	0	0	0	0	0	0	0
Typography	16	23	39	18	83	2	10	0	1	1	1	0	1.1e+03	0	0	0	0	0	0	0	0	0	0	0
Checked	2	1	1	0	0	0	0	0	0	0	0	0	0	16	0	0	0	0	0	0	0	0	0	0
Conversational	15	3	2	1	0	0	0	0	0	0	0	0	0	0	1.1e+02	0	0	0	0	0	0	0	0	0
Superhero	12	0	5	1	0	0	0	0	0	0	0	0	0	0	0	99	0	0	0	0	0	0	0	0
Camouflage	1	0	0	0	0	0	2	1	0	0	0	0	0	0	0	0	16	0	0	0	0	0	0	0
Music	3	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	29	0	0	0	0	0	0
Striped	79	55	6	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	7.8e+02	0	0	0	0	0
Varsity	6	1	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	50	0	0	0	0
People and Places	15	0	6	4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.4e+02	0	0	0
Humour and Comic	18	0	2	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.2e+02	0	0
Self Design	2	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	27	0
Sports	3	5	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	54

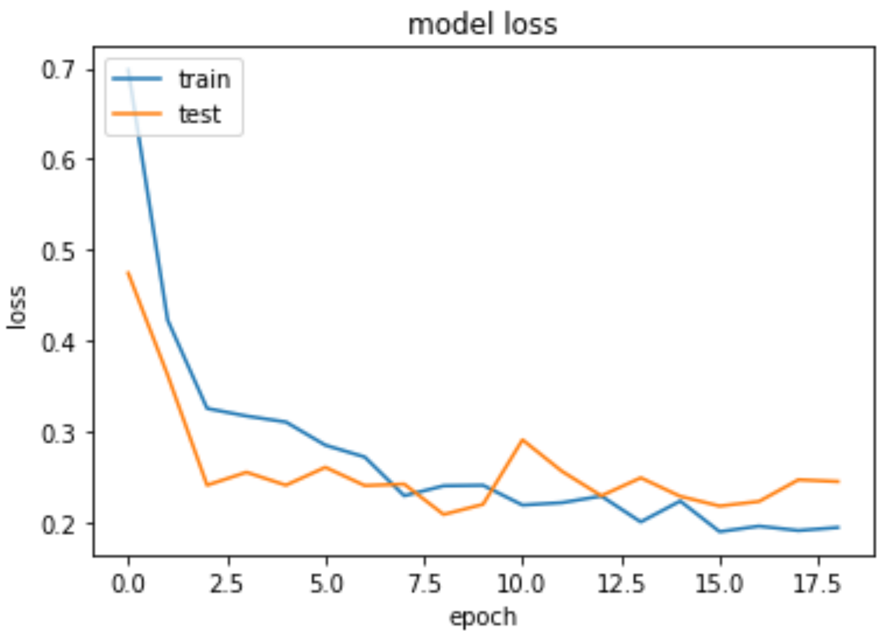
Figure 4 Confusion matrix, without normalisation

Justification

The final results are found stronger than the benchmark result reported earlier. The accuracy of the benchmark model was 4.35%, but it improved to 84.75% accuracy with two models' development.

The following two graphs show the performance of the final model. Accuracy *almost* always increases with each epoch. The loss value indicates how good or bad a particular model is each time the optimization is repeated. Ideally, the loss value is expected to decrease after each or several repetitions.

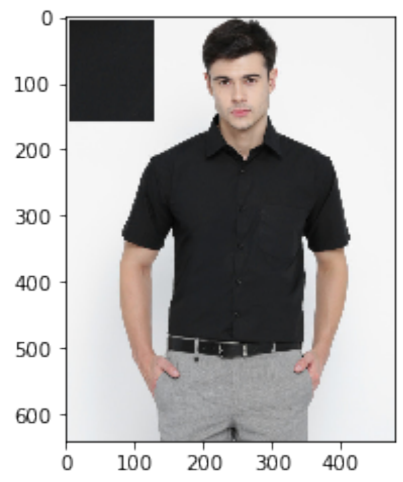




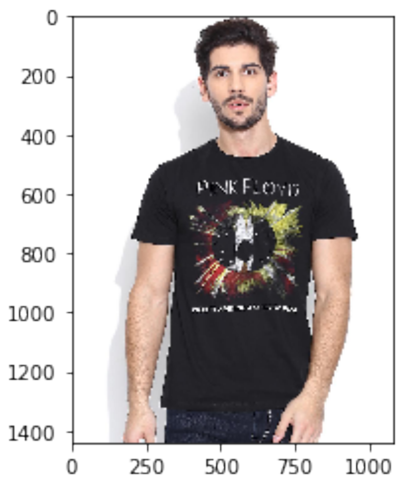
5. Conclusion

Free-Form Visualization

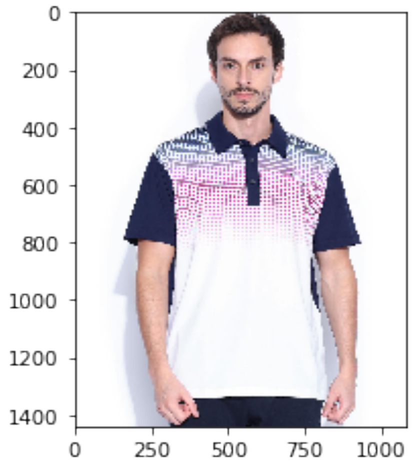
The predicted results of the final model were compared with the actual label values. Let's look at the predictions for the four image data below. It can be seen that *graphic type* is **different** from the actual label value. In other words, **not** all of the following results are successful predictions.



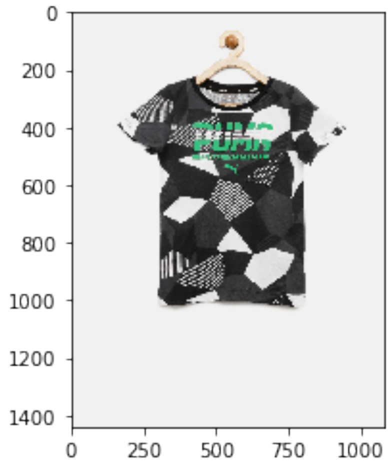
```
I'm sorry, Dave. I'm afraid I can't do that!
(Correct: Shirt)
```



```
That's a t-shirt. Graphic type: Solid
(Incorrect: Music)
```



```
I'm sorry, Dave. I'm afraid I can't do that!
(Incorrect: T-shirt with a Collar)
```



```
I'm sorry, Dave. I'm afraid I can't do that!
(Incorrect: T-shirt on a Hanger)
```

Reflection

For reaching into this end to end solution, I've tried to progressively use models, with incrementing complexity, to classify the images. The random chance presents an exceptionally low bar: setting aside the fact that the classes are slightly imbalanced, a random guess will provide a correct answer roughly 1 in 24 times, which corresponds to an accuracy just above 4%. I've even tried a baseline convolutional model as a good-practice because I wanted to see how the model performs with with a few number of layers only (it heavily underperforms unfortunately with an accuracy of 18.90%). To come to the point of using transfer learning, I had to extract the bottleneck features for the the VGG16 and ResNet50 models first, and then experiment with running different versions top layers on the features. Between the two aforementioned models, I saw significant improvement in the results, and so I decided to use the ResNet50 architecture rather than VGG16 because of the computational costs involved.

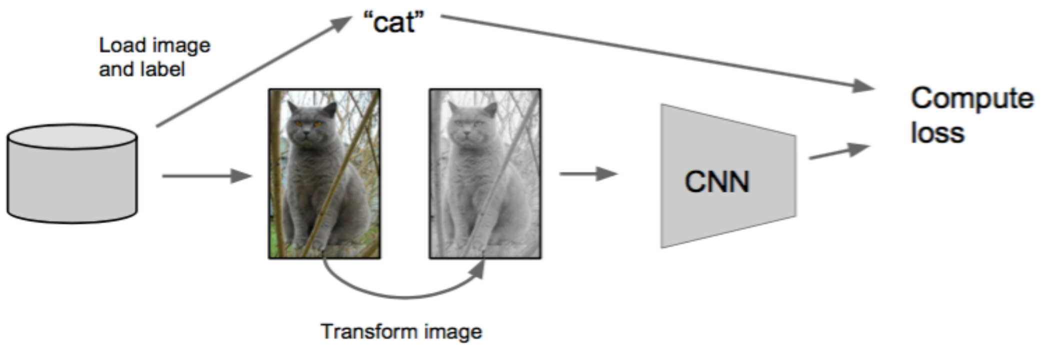
Even if the quality of this dataset is quite high, given it shows the humans exhibiting different styles of t-shirts, and also t-shirts hanging on hangers, I'm uncertain if this dataset is a comprehensive representation of the catalogue the system would face in real life because of small changes such as t-shirt lying in a pile of items, can easily offset the model as the background will be changed. I believe a boundary box approach that'd be able to detect the t-shirt in the image via object detection, crop the image to zoom into the t-shirt and then classify it will have a better chance.

The most difficult part for me was to get the experiments running on my local machine. Higher computational time results in lower number of experiments when it comes to neural networks, specially when I'm just figuring out what to do as it's my first experience with deep learning. However, as I feel more confident in my skills in using deep learning now, I'll definitely try to seek more computational resources from now on. I'm grateful for the effort that was put in accumulating the dataset, as I find it the most unattractive step. ☺ I find myself more engaged in the analytical or logical thinking, which actually makes the problem worthier to solve.

Improvement

Image Augmentation

Image augmentation, or data augmentation, is a method used to improve CNN performance. If you move all the pixels in the cat image one space to the right, it looks the same in the human eye. However, because the computer expresses and recognizes the image as a pixel vector, the cat image moved by one pixel is recognized as different from the original image. To solve this problem, we use data augmentation.



To summarize again, data augmentation is a method of changing a pixel without changing the label of the image, and proceeds with learning using the modified data. Also, almost all network models to date have used data augmentation, which is a very common method.

Change Optimizer

I didn’t get much time and space to experiment with different optimizers such as SGD (Stochastic Gradient Descent), or Adam (Adaptive Moment Estimation). RMSprop is an unpublished adaptive learning rate, but yet widely used across neural networks. I still have to form a base, and a better understanding for myself to learn, and experiment more with the available optimizers.

6. References

- http://cs231n.github.io/transfer-learning/ "CS231, Andrej Karpathy's overview on Transfer Learning"
- http://www.fast.ai/ "Fast Ai MOOC, understanding transfer learning"
- https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html "Building classification models using very little data - Francois Chollet"