Machine Learning Engineer Nanodegree

Capstone Project

Deepak Batra
8th August 2018

I. Definition

BACKGROUND

Myntra is India's largest fashion e-commerce company. A variety of products are listed on Myntra.com **web/Myntra's mobile application** and bought by the customers using the platforms. A customer purchases a wide variety of clothing, be it shirts, t-shirts, jeans etc. Product (clothing) attributes influences a customer's decision process; be it checks on the shirts, patterns on the jeans or graphics on the t-shirts. To make sure that the right inventory is available and the right products are personalised as per the customer's intent/ buying history Myntra needs to extract attributes from the clothing's images and divide products into categories which can then be linked to customers and therefore the content can be personalised per customer. Also, this would help in inventory management as the products which are high on customer intent will be bought more and therefore increasing saleability on the platform.

PROBLEM STATEMENT

As discussed above, deciphering attributes from an image of clothing is a problem that can help the business in many ways than one. But categorising all types of clothing requires a big train/test dataset and therefore a smaller **classification problem** is chosen which is to identify and categorise the graphic type of a **t-shirt** type clothing only. However, the same can be extended to other types of clothings be it **shirts**, **jeans** etc.

To define the problem, a **t-shirt** is going to be categorised into 24 different pre-defined categories. The categorising model will be used in cataloguing new t-shirts in Myntra by extracting graphic-type attribute and therefore a customer can search/filter based on this attribute from the mobile app/web client. The model takes a set of images and predicts its category-type. The datasets are divided into **test** and **train** data and the **accuracy metric** of the data is going to be the **percentage prediction** in images in the test-data.

DATASETS AND INPUTS

The data is divided into two datasets namely **test** and **train** dataset. The features being used are as follows:

`Brand, Article-type, Gender, Colour, Image, [Graphic Type[1]]`

Brand is an integral part of the features as specific brands are known for their articles/ graphic-types.

Article-type (ex. t-shirt/shirt etc) is included so as to extend this model to predict the categories for shirts/jeans later on.

Gender is a again a determining factor in identifying the graphic of type

Image from which the graphic-type is extracted and category is found.

To segregate the images into different folders based on graphic types a small JAVA program was written which downloaded the images from their respective URLs in the original dataset and were put into specific folders with folder names as *graphic_type_id.graphic_category*. The number of images in the train + validation set were around **8K** and in test were around **4K**. The dataset was cut down from 1 lakh images to this size as processing these images would require either huge CPU time or a good GPU. Because of scarcity of both the resources the decision was made to cut down the dataset.

In the recent times deep learning has been very successful in developing algorithms/ models which work on images and therefore it was the first choice to solve this problem., especially **Convolutional Neural Nets** have been quite successful. The solution (in the submitted notebook) tries to build three models. One which uses multiple Conv2D layers and then a Dense layer and a final 24 neuron output layer and others which make use of transfer learning and build upon it. Specifically, **ResNet50's** bottleneck features were calculated and then are fed to a **Sequential** network which are acceded by:

Model 1:

1. **Dense** layer with 512 neurons and **relu** as the activation function
2. **Dropout** layer with **40%** dropout
3. **Dense** layer with 24 neurons and **softmax** as the activation function

[1] *derived column*

Model 2:

1. **Global Average Pooling 2D** layer
2. **Dense** layer with 24 neurons and **softmax** as the activation function

METRICS

The accuracy metric for this model will be the **percentage** of total images that the model guesses right once it is trained on the **train** dataset. To give it a formula:
where X = ∑ number of attributes guessed correctly,
T = ∑ records in the dataset

$$P = (X \div T) \times 100$$

A metric like F1 score cannot be chosen as it is mainly used in binary classifications. Also, as the graphic types are independent of each other there's no concept of relevance and therefore accuracy metrics based on precision and recall doesn't make sense.

The loss function used for this project is multi-class algorithm loss also known as **categorical cross-entropy** as a loss function whose formula is defined in Figure 1. The **cross entropy** between two probability distributions p and q over the same underlying set of events measures the average number of bits needed to identify an event drawn from the set, if a coding scheme is used that is optimised for an "unnatural" probability distribution q, rather than the "true" distribution p. And accordingly each image is labelled

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

with one true class with a probability of 1 and a set of predicted probabilities emitted by the model.

Figure 1: ***n*** *is the number of images in the test-set.* ***m*** *is the number of image class labels,* ***y_{ij}*** *is 1 if observation* ***i*** *belongs to class* ***j*** *and* ***0*** *otherwise, and* ***p_{ij}*** *is the predicted probability that observation* ***i*** *belongs to class* ***j****. A perfect classifier will have the log-loss of 0.*

II. Analysis

DATA EXPLORATION

As described in the section above images are fed to a CNN network which predicts their graphic types. Each input image is of 1080 x 1440 pixels with 72 pixels/inch DPI. The images belong to one of the 24 categories of graphic-types defined. Also, worth mentioning is that the images not only contain t-shirts but have different lighting conditions, models posing with T-Shirts and some of the photos are taken from different angles as well, which makes it even more difficult to have a good prediction model build on it. Although, a model trained on these type of images will be robust as these are real-life images rather than canned ones and therefore if trained with the full lot i.e. 1 lakh images, this model can turn out to be of really good use for production grade systems and services. The sizes for the training, validation and test sets are 6676, 1580 and 3560 respectively. The input images are of the specified dimensions and the data is fully labelled. Figure 2 shows the training set distribution.

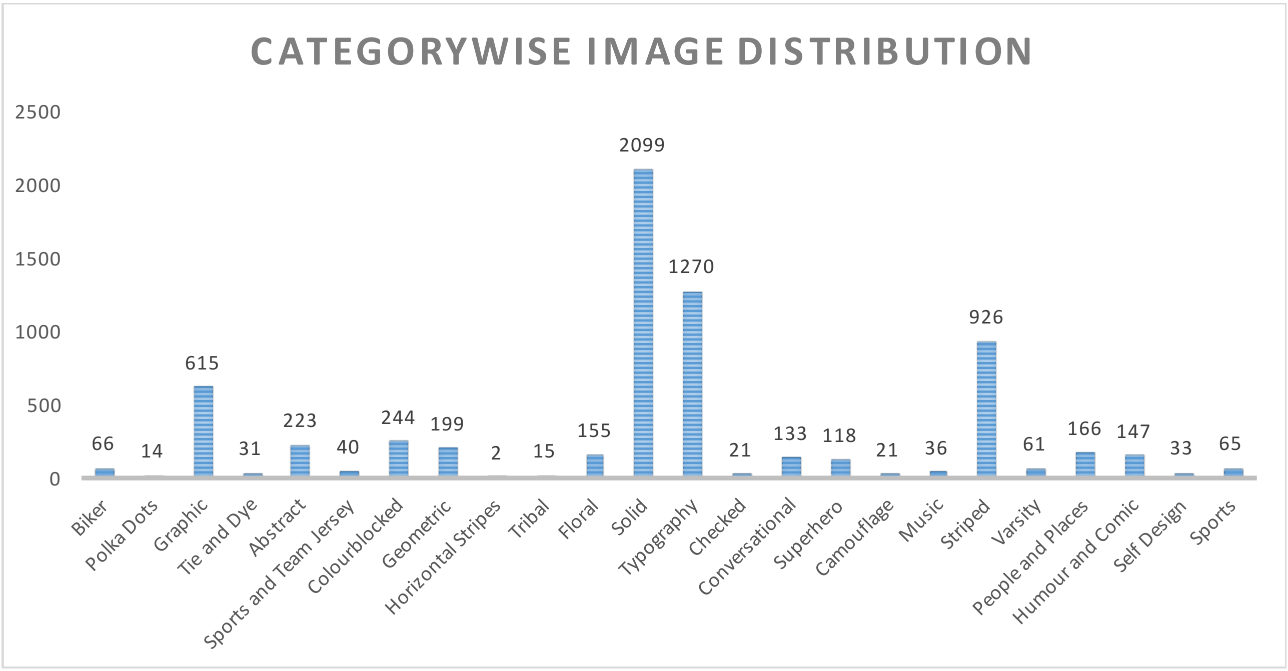


***Figure 2**: Histogram showing the category-wise image distribution of the training set.*

Features

The pixels in the image serve as features for the model. Each pixel of a coloured image has an **Red Green Blue** component and therefore the input is a 3D array which is pre-processed in a specific form and fed to the CNN thereafter.

Image Labelling

Each image is labelled by a graphic-type and as mentioned above a small JAVA program was written to download the images and move them to specific directories with the range of 0-23 and the respective graphic types. Figure 3 below shows some images being used and their graphic-type label as well.

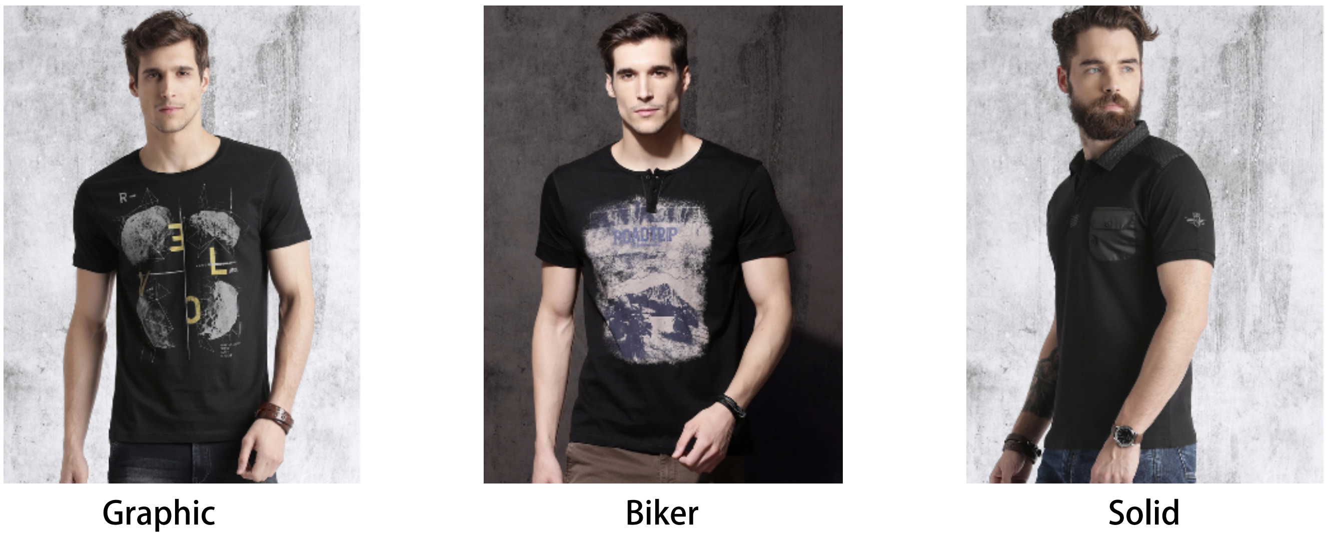


Figure 3: The first image is labelled as ***Graphic*** *image. The second one as* ***Biker*** and the third one is ***Solid.*** *The labels are represent 3 of the 24 categories of t-shirt images present in the dataset.*

Image Distribution

A thing to note here is that the images are not distributed evenly in all the 24 categories. One of the primary reasons for that is **Myntra** being an fashion e-commerce marketplace stocks mainly the categories which are famous and are in demand, and therefore the skewness of the dataset as the whole dataset is sourced from our own catalog here at **Myntra**.

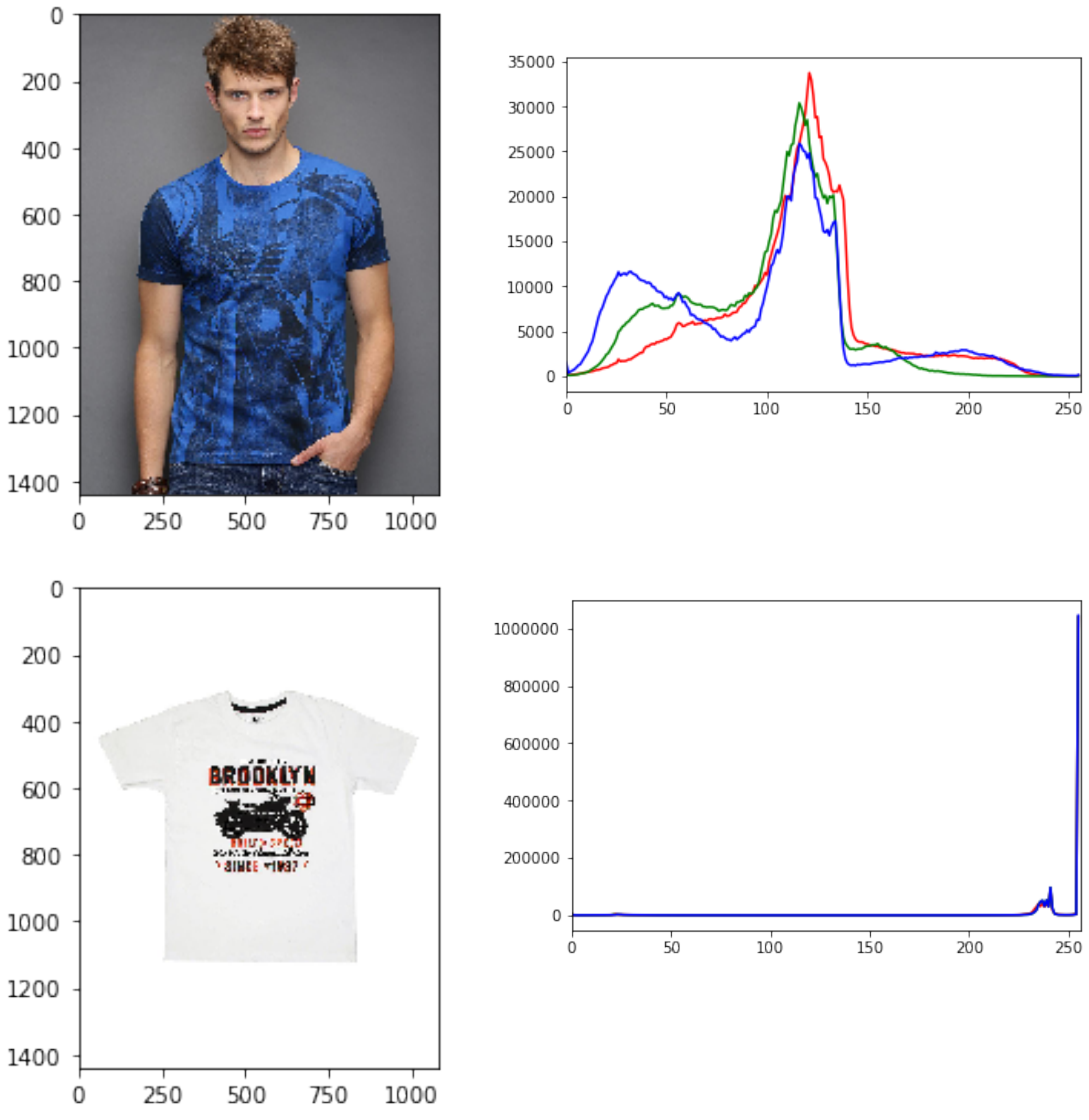
EXPLORATORY VISUALISATION

Image classification has been a research problem since a long time. Various techniques were developed to find similar images and categorise them accordingly. A few techniques are discussed below.

- **Key-point Matching** is a standard approach in computer vision. It tries to pick say 100 important points in an image preferably in corners and edges. These days **SIFT key-points** are the most popular as these can match images with different scale, colour and lighting conditions. However their implementation requires some background knowledge to implement, and the algorithm is slow.

- **Histogram Method** uses elementary image processing, and is potentially faster than the key-point matching, and is easy to implement but it is less robust as it fails to match on scaled, rotated, or dis-coloured images. The idea is to build feature histograms for each image and the images with most matching histograms are predicted to belong to the same category. Re-scaled, rotated, or discoloured images can fail with this method, but small changes like cropping won't break the algorithm. The figure below (Figure 4) shows the RGB histograms of two randomly selected images from the train-set.

***Figure 4**: Sample images and their respective RGB histograms*



Though the above algorithms can help determine and classify images but both the algorithms have their downfalls. On one hand, **Key-point Matching** is a widely used algorithm but it is slow and needs some background knowledge, on the other **Histogram based classification** can be buggy given the kind of images included in the dataset of this project as the image-set contains images of T-shirts with models, under different lighting conditions, with different t-shirt image scale and different angles.

ALGORITHM AND TECHNIQUES

As the techniques described above may not solve the problem statement we have, deep learning is a field worth exploring as it has been able to advance a lot in image specific algorithms. And therefore Convolutional networks are explored and tried out as a solution to the problem. Some of the theory about convolutional network is given below:

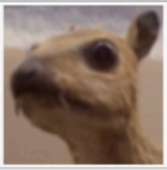
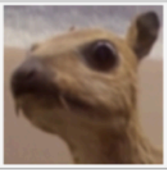
Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Figure 5: *Hierarchal feature extraction in stages*

Convolution:

The primary purpose of Convolution in case of a CNN is to extract features from the input image. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data. In CNN terminology, the 3×3 matrix is called a '**filter**' or 'kernel' or 'feature detector' and the matrix formed by sliding the filter over the image and computing the dot product is called the ***Convolved Feature*** or ***Activation Map*** or the ***Feature Map***. It is important to note that filters acts as feature detectors from the original input image. The evolution of layers or filters can be visually explained by the images in Figure 5. **Conv2D layer** is utilised in the basic CNN build for this project.

Pooling:

Spatial Pooling (also called subsampling or downsampling) reduces the dimensionality of each feature map but retains the most important information. Spatial Pooling can be of different types: Max, Average, Sum etc. In case of **Max Pooling**, we define a spatial neighbourhood (for example, a 2×2 window) and take the largest element from the rectified feature map within that window. Instead of taking the largest element we could also take the average (**Average Pooling**) or sum of all elements in that window. The figure below describes **Max Pooling:**

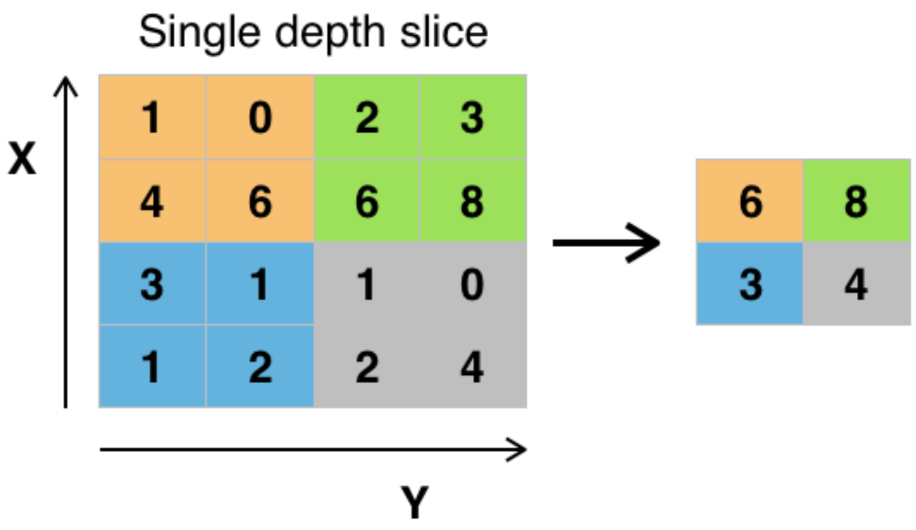


Figure 6: *MaxPooling 2x2 in action*

MaxPooling2D, AveragePooling2D and **GlobalAveragePooling** are the layers used in the basic and the final CNN built.

Dropouts:

Dropout is a regularisation technique, which aims to reduce the complexity of the model with the goal to prevent overfitting. It attaches a probability to each neuron to be active or in-active. If a neuron is in-active in a particular batch then the probability of that neuron firing and hence contributing to the output becomes zero.

Flatten:

As the output of all the above layers will be a multi-dimensional vector, we need to flatten out the array to make it a 1D vector so that it can be fed to a dense or an output layer as the output is 1D one-hot encoded vector in our case.

Dense:

A dense layer is just a regular layer of neurons in a neural network. Each neuron receives input from all the neurons in the previous layer, thus densely connected. The layer has a weight matrix **W,** a bias vector **b,** and the activations of previous layer **a.**

Optimisers:

Adaptive Gradient Algorithm (AdaGrad) that maintains a per-parameter learning rate that improves performance on problems with sparse gradients (e.g. natural language and computer vision problems).

Root Mean Square Propagation (RMSProp) that also maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight (e.g. how quickly it is changing). This means the algorithm does well on online and non-stationary problems (e.g. noisy).

Adaptive Moment Estimation (Adam) realises the benefits of both AdaGrad and RMSProp. Instead of adapting the parameter learning rates based on the average first moment (the mean) as in RMSProp, Adam also makes use of the average of the second moments of the gradients (the uncentered variance). Specifically, the algorithm calculates an exponential moving average of the gradient and the squared gradient, and the parameters β_1 and β_2 control the decay rates of these moving averages.

All the flavours of the architecture of CNN explored in this project to predict the images use **Adam** as the optimiser.

Activation Function:

Consider a neuron which holds this value:

$$\boldsymbol{\gamma} = \sum \textit{(weights * input) + bias}$$

The value of $\boldsymbol{\gamma}$ can be anything between $(-\infty\ ,\ +\infty)$ and therefore to check whether the output value produced fires a neuron hence makes this neuron contribute to the output an activation function is needed.

ReLu is a non-linear activation function as shown below with its graph. It gives an output x if x is positive and 0 otherwise. Any combination of ReLu function is also non-linear.

$$A(x) = max(0,x)$$

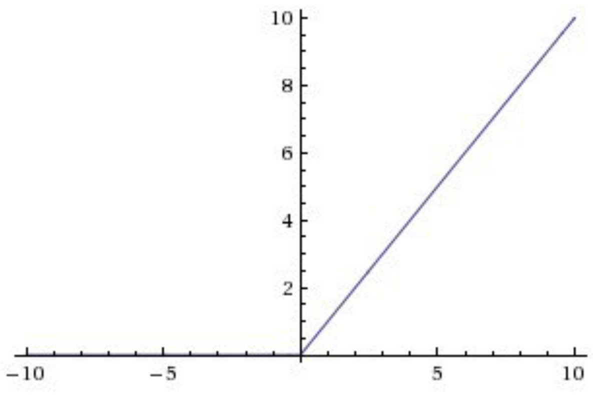


Figure 7: *Graph of a ReLu function*

Softmax is generally used in the output layer. It converts every output to been in the range of 0 and 1, just like the Sigmoid function. But it divides each output such that the total sum of the outputs is equal to 1.

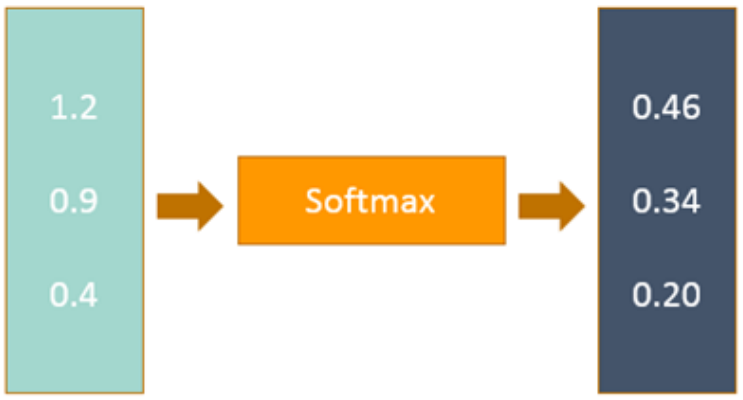


Figure 8: *Softmax function*

The CNNs developed in this project use both **ReLu** and **Softmax** as the activation functions.

Training:

The training process involves feeding the input to the **neural network** to find optimal **γ** value for each neuron. This value is then fed to an **activation function** which then activates or fires one or more neurons. These neurons which in-turn are connected to say a **dense layer** then contribute to the weights of the edges connected to it and a prediction is made based on these weights. However the predictions can be wrong which is determined by comparing the one-hot encoded output from this network to the already labelled training data set. Once the error or loss is found out by using different algorithms like **categorical cross-entropy** etc, a technique called **back-propagation** is used to adjust the weights of the whole network. **Optimisers** are used for this process to back-propagate and optimise the weights so that the network converges faster. A typical CNN architecture and its size per layer is shown in the below figure.

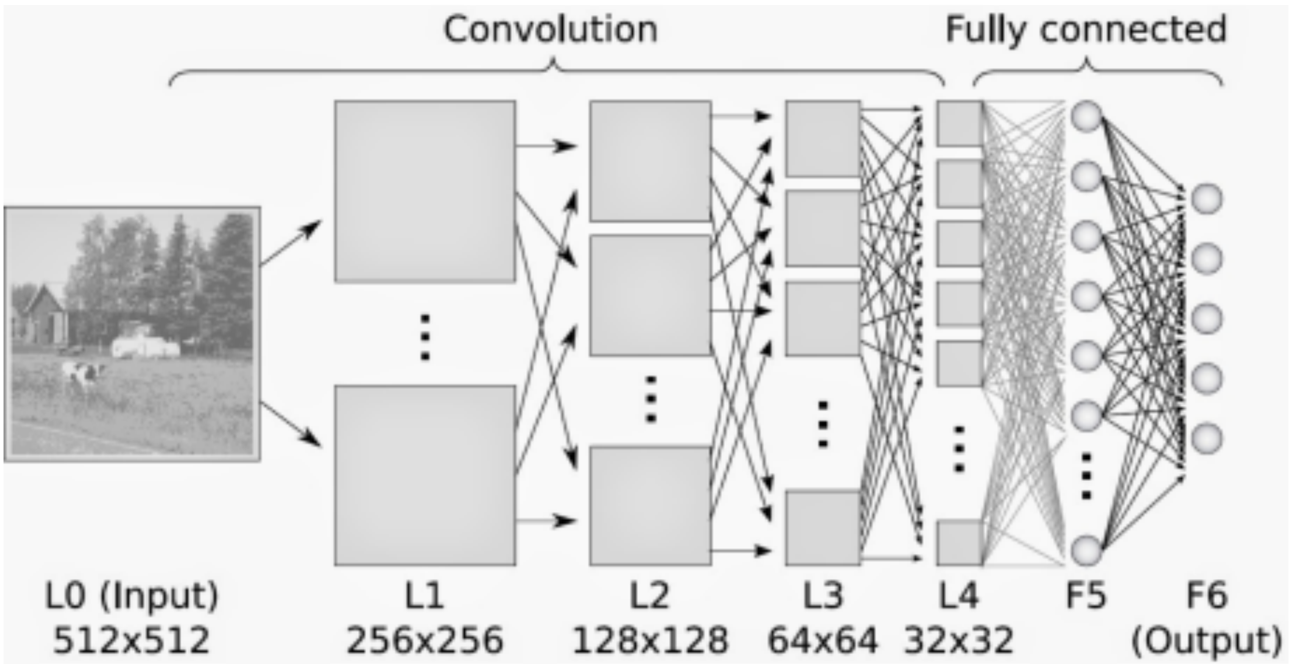


Figure 9: *A typical Convolutional network with convolution layer at the front and fully connected dense layer in the end.*

Transfer Learning:

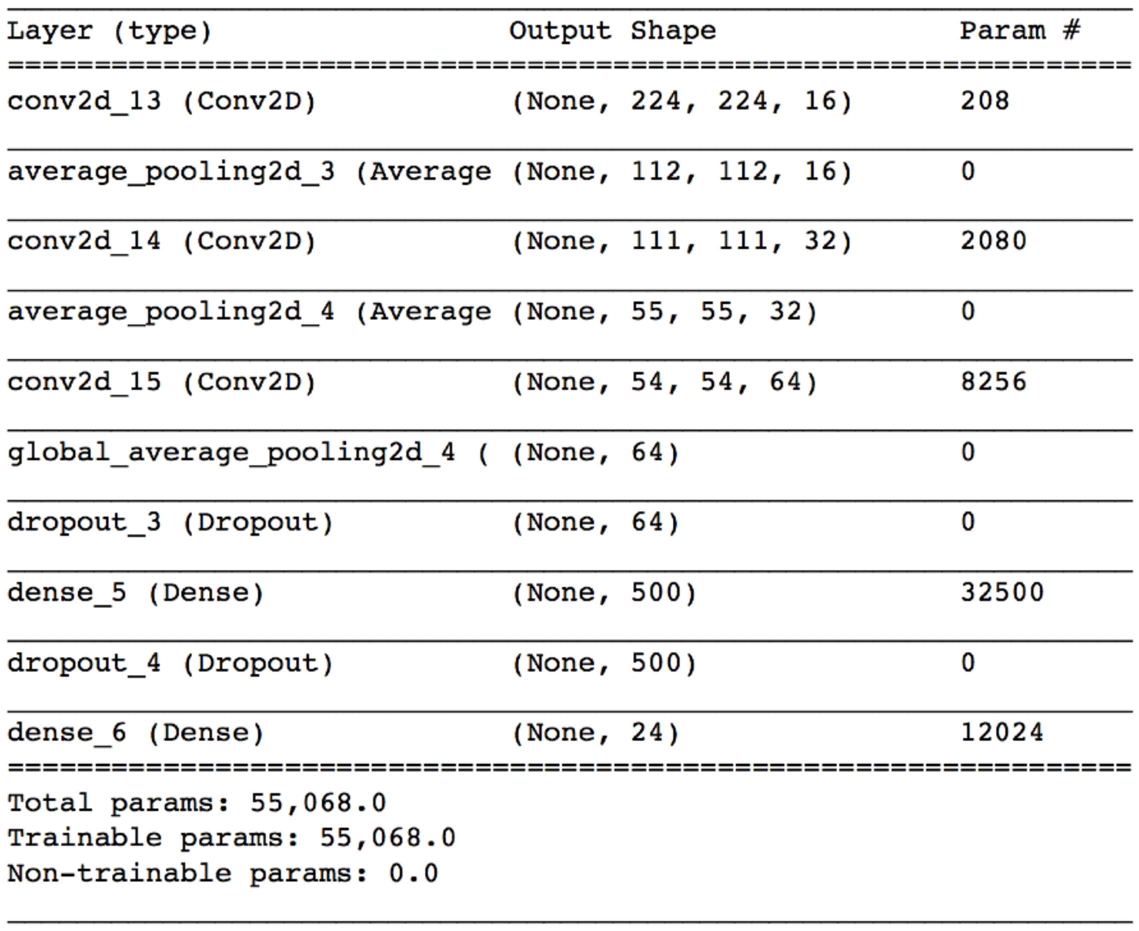
Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task. It is a popular approach in deep learning where pre-trained models are used as the starting point on computer vision and natural language processing tasks given the vast compute and time resources required to develop neural network models on these problems and from the huge jumps in skill that they provide on related problems.

In this project **ResNet50** is the model pre-trained on **ImageNet** dataset which contains over 10 million URLs, each linking to an image containing an object from one of 1000 categories. As we have seen in the above section, the first few layers of the CNN create the filters or the feature maps, however the last layers are generally dense layers which try to predict the output of the network and this introduces the concept of residual learning. Residual learning tries to short-circuit layers i.e. layer say L_n's output is fed directly to say a layer L_{n+m} in the network. As we have features maps extracted for a huge corpus of images it is easier to adjust the weights of the existing set by running through the newer set of images by the CNN after removing the dense layers on the top. Once the bottleneck features are calculated using this technique, we can add the final dense/other layers as and when required.

Training a pre-trained deep network this way is a lot easier and quicker. It also solves the accuracy problem in case the training set is small and is not representative of the entire gamut of categories one wants to predict. On the more, it takes less time, resources to train and predict as compared to a built-from-scratch deep network.

Benchmark:

A **Basic CNN** model is developed which consists of three convolutional layers. Each one followed by a pooling layer. AveragePooling is used after the first Conv layer. MaxPooling is used after the second and GlobalAveragePooling is used after the third. This is followed by a Dense layer with 500 neurons, a dropout layer with 40% drop and the final dense output layer which has 24 output neurons each representing a graphic type. The architecture and the hyper-parameters at each layer is given below.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_13 (Conv2D)           (None, 224, 224, 16)      208       
_________________________________________________________________
average_pooling2d_3 (Average (None, 112, 112, 16)      0         
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 111, 111, 32)      2080      
_________________________________________________________________
average_pooling2d_4 (Average (None, 55, 55, 32)        0         
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 54, 54, 64)        8256      
_________________________________________________________________
global_average_pooling2d_4 ( (None, 64)                0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_5 (Dense)              (None, 500)               32500     
_________________________________________________________________
dropout_4 (Dropout)          (None, 500)               0         
_________________________________________________________________
dense_6 (Dense)              (None, 24)                12024     
=================================================================
Total params: 55,068.0
Trainable params: 55,068.0
Non-trainable params: 0.0
_________________________________________________________________
```

Figure 10: *Architecture of the basic CNN developed as a benchmark model for the project. The image shows the shape and parameters at each layer and its type as well.*

The accuracy of the above defined network architecture on the test set came out to be **18.9045** %.

III. Methodology

DATA PREPROCESSING

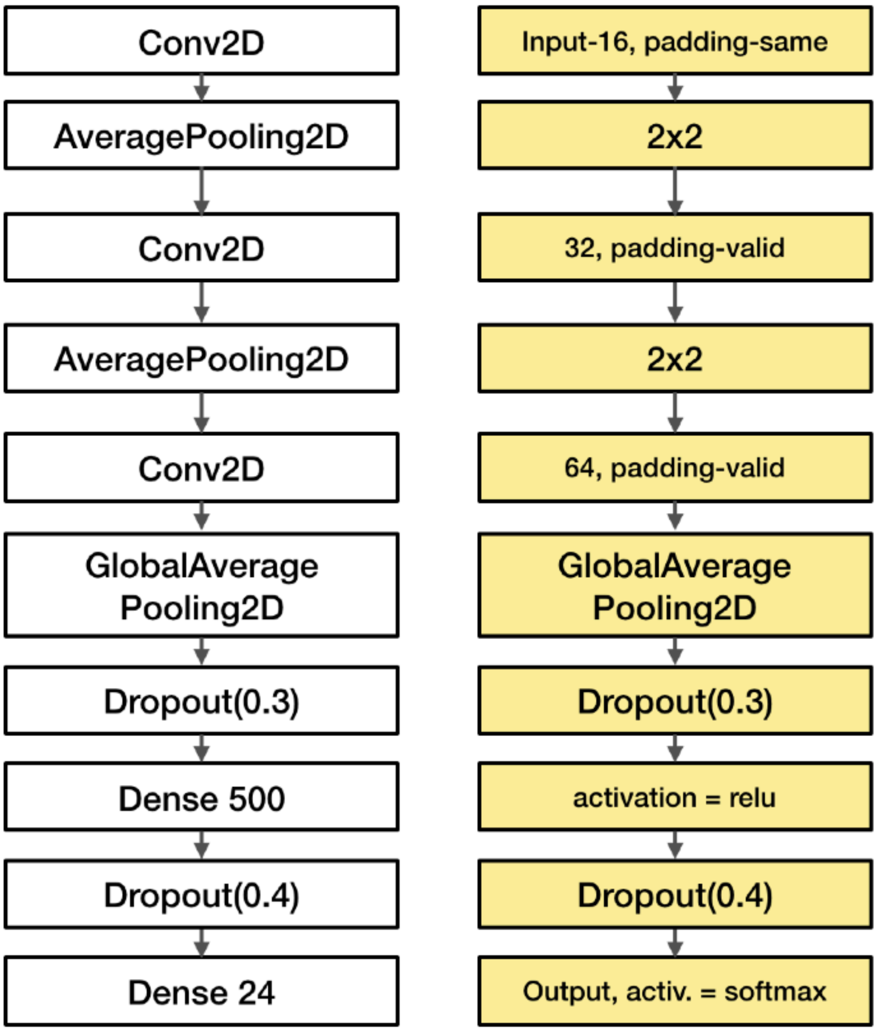
The images are all converted to 3D tensors each dimension corresponding to a colour component i.e. RGB which are then appended by expanded by a dummy number 1 to create a 4D tensor which is further processed before being fed to the ResNet50 model. The pre-processing step converts the RGB image tensor to BGR tensor i.e. re-orders the channels in the given tensor. Then the pixels are normalised by subtracting the mean pixel value *[103.939, 116.779, 123.68]* in RGB calculated from all images present in ImageNet) and then division by the max-value of a pixel which is *[255]*. Normalisation is a necessary step as it somewhat helps in reducing the noise introduced by different ratios of the image size to the t-shirt section size in a particular image. Also, different lighting conditions and other deformities are reduced to a good extent otherwise higher R/G/B values may contribute more to the weights and therefore the feature maps can be completely skewed.

IMPLEMENTATION

ConvNet:

The recommended size of **filters** for input layers in CNNs is **16** with a **kernel size of 2.** The idea is to increase the numbers of filters in subsequent layer. And therefore the three **Conv2D** layers of filter size 16, 32 and 64 are cascaded with pooling layers in between. In between these layers an **AveragePooling2D** layer of size **2x2** is used. Its function is to progressively reduce the spatial size of the representation to reduce the amount of computation and hence to control overfitting. I tried using a **MaxPooling2D** layer instead of this, however the accuracy of the model reduced by 1-2%. Instead of a **Flatten** layer a GAP layer is used which is followed by a **Dense** layer of 500 nodes. **GAP** was used as it gave better performance than **GMP** or **Flatten** layer. The activation functions used were **ReLu** and **Softmax. Dropout** percentages of **30%** and **40%** optimised the runtime of the algorithm without any decrease in accuracy. The first model built didn't have any dropouts but to prevent overfitting I added these, however this didn't have any impact on the accuracy. Loss function was **categorical cross-entropy** and the optimiser used was **Adam** because of its fast convergence rate. **RMSProp** decreased the overall accuracy as the number of epochs were kept less due to limited compute power and hence was discarded. The basic Convolutional Neural Net architecture developed hereby is shown in Figure 11.

Figure 11: Architecture of the basic CNN developed as a benchmark model for the project. The image below shows the shape and parameters at each layer and its type as well.

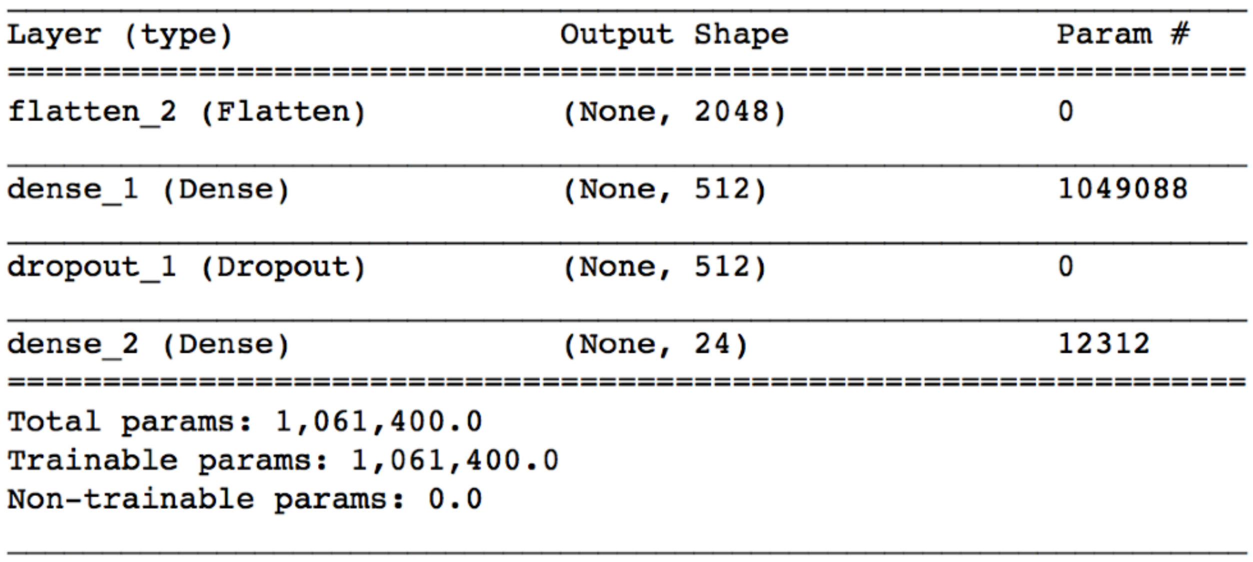


Batch size is kept at **20** and the number of **epochs** is kept at **5**.

Transfer Learning based models:

Model Architecture 1:

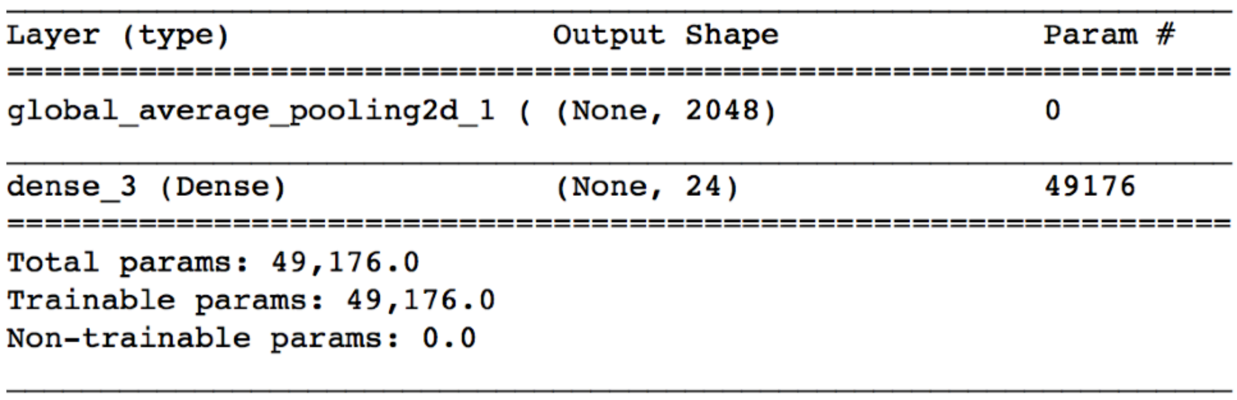
The first model that was developed is as given in the Figure 12. Bottleneck features were calculated for ResNet50 model and were loaded. A **Flatten** layer was used to convert the multi-dimensional input to a one-dimensional array. It was succeeded by a **Dense** layer with 500 neurons with activation function as **ReLu** and a **dropout** of 40% to prevent overfitting. And then a final output layer of 24 neurons with activation function as **Softmax** was used. The model was able to achieve an accuracy of **29.09%**.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten_2 (Flatten)          (None, 2048)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 512)               1049088   
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 24)                12312     
=================================================================
Total params: 1,061,400.0
Trainable params: 1,061,400.0
Non-trainable params: 0.0
_________________________________________________________________
```

Figure 12: *Architecture of the first transfer learning (ResNet50) based CNN developed for the project.*

Model Architecture 2:
I tried to optimise the model further by adding multiple dense layers and multiple dropout layers. However, with each dense layer added, the epoch time increased and the accuracy dropped either by a few percentage points or more. So, I tried removing all the dense layers except the final output layer and came up with the architecture given below.
I replaced the **Flatten** layer with a **GAP** layer as well. To my surprise the accuracy increased by greater than **2%** and came out to be **31.12%**. This can very well be attributed to overfitting because of the added dense layers and therefore the model shown in Figure 13 is the final model.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
global_average_pooling2d_1 ( (None, 2048)              0         
_________________________________________________________________
dense_3 (Dense)              (None, 24)                49176     
=================================================================
Total params: 49,176.0
Trainable params: 49,176.0
Non-trainable params: 0.0
_________________________________________________________________
```

Figure 13: *Architecture of the final and best accuracy transfer learning (ResNet50) based CNN developed for the project.*

Both the models used the **loss function as categorical cross-entropy**, the **optimiser as Adam**, **batch size as 20 and epochs as 20** as well. The loss function, optimiser and the batch size were kept the same for all models to have **parity** so that these parameters have no effect when comparing their performance. The activation function used in the output layer of the both the models was **Softmax.**

Stack Used:

- Python 3.5
- NumPy
- Pandas
- Keras
- TensorFlow
- matplotlib
- Jupyter

The final model which is able to predict the graphic type with **31.12 %** on the test-set is built using the technique of transfer learning on ResNet50 ConvNet. Transfer learning seems to have solved the problem of less data and has improved the accuracy of the model by a huge margin.

REFINEMENT

The basic model that was developed took a lot of time to train and the accuracy percentage of **18.90%** was also not upto the mark. However using the technique of transfer learning, another model based on ResNet50 was developed which had better accuracy **(29.09%)** and took lesser time to train as well. This was a big improvement on the basic model. However, it took me some good time to refine the model further with a lot of brainstorming about adding more dense layers, increasing/decreasing dropout percentages. Finally I was able to get an accuracy of **31.12%** by removing the overfitting problem by removing the dense layers from the earlier developed model and by using just a single output layer as explained above.

Another improvement called **K-Fold cross validation** suggested in the review was utilised to prevent over-summing and solve the problem of insufficient data volume and increase the statistical reliability of the CNN classifier performance and is explained in the later sections.

As explained earlier, the original image-set consisted of around 1 lakh images but due to resource constraints I had to cut down on the number of images in the input dataset. Having more images in the train, valid and test sets can further improved accuracy as with less images, neural nets tend to remember the data/images used. This is a typical problem is generally solved by throwing more data in. However due to resource constraints I was only able to use my Mac's GPU to run these experiments. Another optimisation that would help is if the image can be cropped and only the relevant portion i.e. the t-shirt area is fed to the CNN downstream which would've reduced noise and would've led to even better accuracy.

Image augmentation wasn't done because of similar reasons, and could've helped improve the accuracy. Higher number of **epochs** and adding a few **dense layers** at the end with the right drop-out would've led to a more refined model.

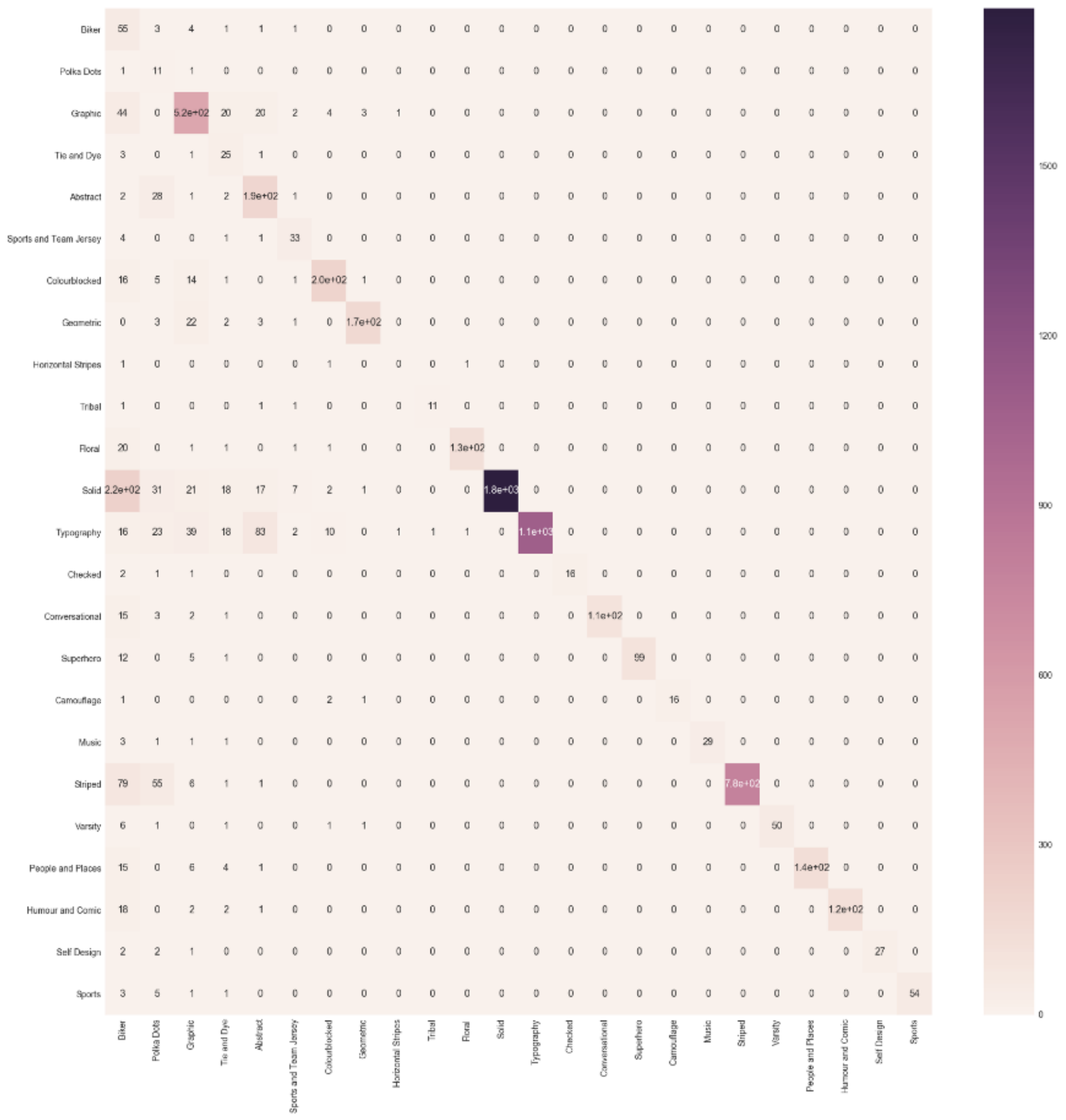
ACCURACIES

Basic CNN's accuracy came out to be **18.9045** % on the test-set
ResNet50 transfer learning based **CNN's accuracy** came out to be **31.1151** %

IV. Results

MODEL EVALUATION AND VALIDATION

By taking a pre-trained ResNet50 model and removing the top level fully connected layers and adding our layers on top did increase the test-set accuracy to **31.12%** is a good result for the start. This also resulted in lesser time and resource utilisation as only the new layers added on the top have to be trained and the weights of earlier layers in the network are loaded and kept the same. These weights are called bottleneck features which were adjusted by training the existing ResNet50 network pre-trained on ImageNet on our train-set. The non - normalised plot of the confusion matrix is as given below. As evident from the confusion-matrix the model is really good with predicting solid and typography categories because as mentioned earlier the data is sourced from Myntra's



	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	18	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 14: *Non normalised Confusion matrix*

own catalogue which is rich in t-shirts belonging to these categories. As these t-shirt designs sell more on our platform therefore the percentage of these designs are much more as compared to others. Also, the maximum number of images in the train, valid and test dataset also belong to the same categories, the model is a little biased towards that.

A modified version of **K-Fold cross validation** was used with transfer learning as the bottleneck features were already extracted and I couldn't find a way to merge train and valid features and find the splits in them. Rather than using library function for generating k splits, it was done as shown below: (boiler plate code removed for brevity)

```
splits = [3, 5, 7, 10]
for k in splits:
    # create the model
    # compile the model
    split_val = len(train_feat)/10
    valrows=([x for x in range(split_val*(k-3),split_val*(k))])
    trainrows=([x for x in train_feat if x not in valrows])
    model.fit(train_feat[trainrows,:], train_targets[trainrows,:]…,
    validation_data=(train_feat[valrows,:], y_train[valrows,:] …)
```

Figure 15: *Pseudo code to simulate k-fold cross validation with transfer-learning*

And here are the stats from the 4 runs:

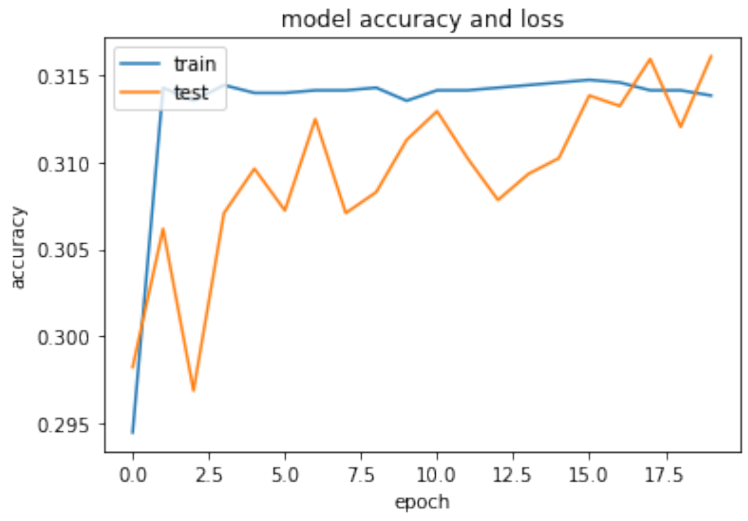
Table showing variance and mean accuracy for various k's

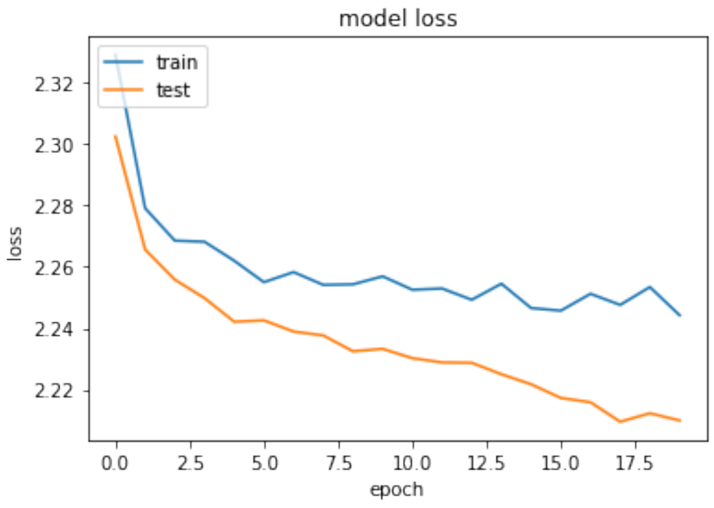
K Value	Variance	Mean Accuracy
3	0.0001860592799	22.14%
5	0.0002406620667	25.73%
7	0.0004088160460	29.17%
10	0.0004858375812	31.12%

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 though the variance remains almost the same throughout. Therefore, k-fold CV when k is 10 is the most robust model.

JUSTIFICATION

The results from the transfer learned model were found to be satisfactory with a higher accuracy as compared to the basic CNN developed as the benchmark. The following plots of accuracy and loss conform to the same hypothesis. Also, the k-fold CV results confirm that the model developed is robust and it is difficult to fool the model.

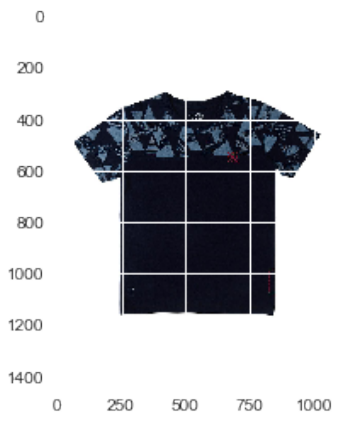




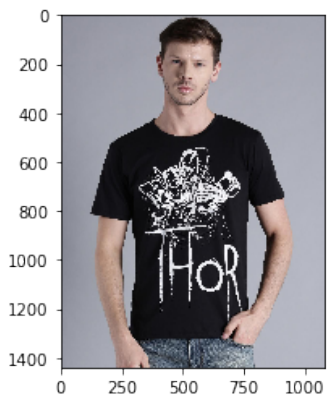
V. Conclusion

FREE FORM VISUALISATION

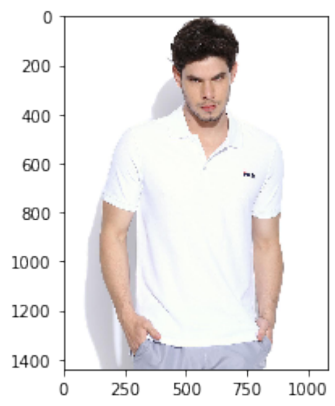
A few sample images were fed to the T-Shirt detector and the results are as shown below. It can be seen that not all the predictions are correct i.e. the true graphic type label is different from the predicted one.



```
Detected a t-shirt with the graphic
type as Striped
```



```
Detected a t-shirt with the graphic
type as Solid
```



```
Unable to detect t-shirt, can't
predict graphic type as well
```

REFLECTION

An important aspect of the whole project was the data. It was easy to find the data but labelling it was a tedious task. The good part was that a lot of data was labelled and images and their respective graphic types were in abundant. So much so, that due to restricted compute power I had to cut down the number of images from nearly 1 lakh to around 11k. The dataset itself required some pre-processing in terms of downloading and folder structure categorising which was done using a multi-threaded java code.

The next steps were also quite huge in terms of pre-processing the images, building a basic CNN, identifying which and how many layers to use, activation function, loss functions and the epoch as well as the batch size. All this led to tremendous learning in the field of deep learning and reinforcement learning and the resource constraints taught things which ideally are not learned in any other environments.

IMPROVEMENTS

The suggested model uses ResNet50, however other models like VGG16, InceptionV3 and Xception can be used in place of it as they are more complex and have better accuracies as well. Another thing which was described in earlier sections as well is a technique called **Image Augmentation** which can be utilised.

V. References

Keras: https://keras.io
Machine Learning blog: https://machinelearningmastery.com/
Transfer Learning: https://cs231n.github.io/transfer-learning/
K-Fold with Transfer Learning: https://github.com/IoannisNasios/Blog