**Introduction:**

**Project Title: Predicting genre of book**

**Objective:**

Object detection is currently a very popular topic in deep learning. Along with the rapid development in computer vision techniques and algorithms, we could precisely detect objects from images regardless of their relative locations and sizes in real time. We would like to see whether computer could recognize the genres of books from their images.

Intrigued by the overwhelming popularity of Deep Learning and the pace at which it is prevailing, this project depicts an attempt of creating a deep learning model for real time implementation to predict Genre of books. There has already been amazing work since the origin of Artificial Intelligence and Deep Learning which spawned many different models. The main goal of the project is to classify books in to their respective Genres.

**Problem Statement**

Given the images of book covers including both fiction and nonfiction images, we have to predict Genre of books.

There are around 12000 images in total which belong to 5 different categories as follows, our goal is to predict which genre does each image belongs to. "Comics &Graphic Novels", "Cookbooks", "Kids Books", "Mystery Thriller &Suspense", "Science Fiction & Fantasy*"*

**Algorithms used:** ResNet50, Vgg16, and Googlenet

**General Introduction to algorithms:**

**ResNet50:** As deeper networks always outperforms the less deeper ones, residual networks are highly deep neural networks. With a residual block, however, a skip function reduces the number of times a linear function is used to achieve an output. A skip function creates what is known as a residual block.

**Googlenet**: GoogLeNet is a convolutional neural network that is 22 layers deep. You can load a pretrained version of the network trained on either the ImageNet [1] or Places365 [2] [3] data sets. The network trained on ImageNet classifies images into 1000 object categories, such as keyboard, mouse, pencil, and many animals.

**VGG16:** In the VGG 16 Model, the convolution stride was fixed to 1 pixel; the spatial padding of convolutional layer input was such that the spatial resolution was preserved after convolution, i.e. the padding was 1 pixel for 3 × 3 conv. layers. ... Max-pooling was performed over a 2 × 2 pixel window, with stride of 2.

**Steps followed:**

**Data collection:** We have downloaded list of book covers from Amazon Book Store touse them as our data source.

**Image Preprocessing**

To prepare the images for our network, we are resizing them to 224 x 224 and normalizing each color channel by subtracting a mean value and dividing by a standard deviation. We will also augment our training data in this stage. These operations are done using image transforms, which prepare our data for a neural network.

**Data Augmentation**

Because there are a limited number of images in some categories, we can use image augmentation to artificially increase the number of images "seen" by the network. This means for training, we randomly resize and crop the images and flip them horizontally. A different random transformation is applied to each epoch (while training), so the network effectively sees many different versions of the same image. All the data is also converted to Torch Tensors before normalization. The validation and testing data is not augmented but is only resized and normalized. The normalization values are standardized for ImageNet.

**Data Iterators**

To avoid loading all the data into memory at once, we used training DataLoaders. First, we create a dataset object from the image folders, and then we pass these to a DataLoader. At training time, the DataLoader will load the images from disk, apply the transformations, and yield a batch. To train and validation, we'll iterate through all the batches in the respective DataLoader.

One crucial aspect is to shuffle the data before passing it to the network. This means that the ordering of the image categories changes on each pass through the data (one pass through the data is one training epoch).

**Pre-Trained Models for Image Classification**

PyTorch has many pretrained models we can use. All of these models have been trained on Imagenet which consists of millions of images across 1000 categories. With pretrained models, we will freeze the early layers, and replace the classification module with our own.

**Approach**

The approach for using a pre-trained image recognition model is well-established:

* Load in pre-trained weights from a network trained on a large dataset
* Freeze all the weights in the lower (convolutional) layers
* Layers to freeze can be adjusted depending on similarity of task to large training dataset
* Replace the classifier (fully connected) part of the network with a custom classifier
* Number of outputs must be set equal to the number of classes
* Train only the custom classifier (fully connected) layers for the task
* Optimizer model classifier for smaller dataset

**Training function**

For training, we iterate through the train Data Loader, each time passing one batch through the model. The below function trains the network while monitoring a number of different parameters. We train with early stopping on the validation set.

**Inference**

After the model has been trained to the point on no more improvement on the validation data, we need to test it on data it has never seen

This function processes an image path into a PyTorch tensor for predictions. It applies the same transformations as was done to the validation data: cropping (center) and normalizing with means and standard deviations.

**Conclusions:**

We were able to make predictions of book covers with an accuracy of 64% using Resnet model. We also explored different Transfer Learning models to train the data and choose the best model to test our resultant data.