Rishika Korade

Image Classification

Problem Statement

Classification of “charts” dataset into 5 categories as dot line, line, pie, vertical bar and horizontal bar charts from the given datasets using Deep learning approaches.

Abstract

The intention of this project is to classify images of a given charts dataset. To be precise, in the case of this dataset, the images were are organized in folders namely train and test. Subsequently there was a csv file containing all the labels of respective image files. The dataset has 128x128 size RGB (3 channel) images falling into one of the five classes, ‘pie’, ‘dot_line’, ‘line’, hbar_categorical’, ‘vbar_categorical’. The classification had been carried out both using traditional CNN architecture, with further iterations of applying regularization techniques including L1 and dropout along with batch normalization at each layer. The second approach taken was to use pretrained model and training its final layer for our required classification model. Observations were made on the basis of time taken for convergence, impact of data augmentations and regularization and finally evaluated using 1 score and confusion matrix. Gradcam approach was also explored by creating a custom dataset of 20 images created from web and screen snippets.

Dataset

The “charts.zip” data contained 1000 images in train_val folder and the lables were contained in train_val.csv file. The train_val.csv file had 2 columns with “image_index” and “type” as its corresponding class category. The test images were in test folder, while evaluating the model two approached were used one was to split the train_val set into train, validation and test folders for ease of evaluation and another was to use the training set provided for training and validation purpose and using the test to test the model on test images without a label to test against but it was verified intuitively. Custom images were also tested individually post model training. The images were of size 128x128 RGB (3 channel). Upon exploring the dataset for random images, it was seen that it did not have any foreground to the main image other than the white background. The dataset did not have any presence of class imbalance (fig 1), all the 5 classes were balanced with 200 images of each category. As a part of pre-processing and segregating the images in real time from directory the Keras “ImageDataGenerator” module was used for generating training (train_generator) and validation (valid_generator) and test (test_generator) images. The images were rescaled. Further augmentations which may have been applied are discussed ahead.

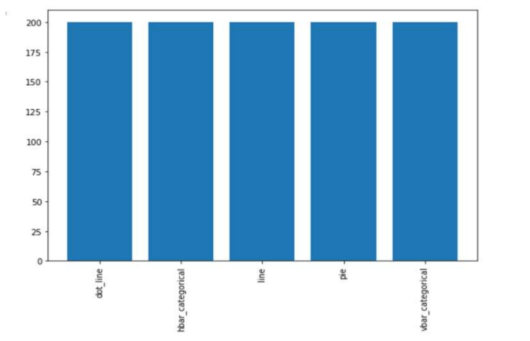
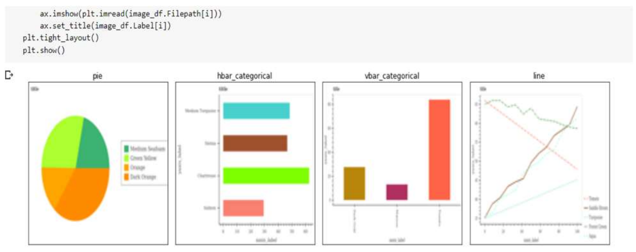


Fig 1. Dataset plot implying the balanced classes



```
    ax.imshow(plt.imread(image_df.Filepath[i]))
    ax.set_title(image_df.Label[i])
plt.tight_layout()
plt.show()
```

Fig 2. Random sample of images from dataset

Implementation

Approach I - 2-layer CNN

Custom Baseline model-

The model type build is Sequential. The model was built layer by layer. First layers is a Conv2D layer. These convolution layers that will deal with input images, which are seen as 2-dimensional matrices. I have used a filter of size 32 with same padding and 2 strides, with relu activation function (swish activation function was also tried but, in this case, it gave the same results). Filter of size 32 was directly applied to the raw input for extracting low level features. This convolutional layer was followed by a max pooling and batch normalization layer in order to normalize the output of previous layers and feed it to consequent layers and avoid overfitting.

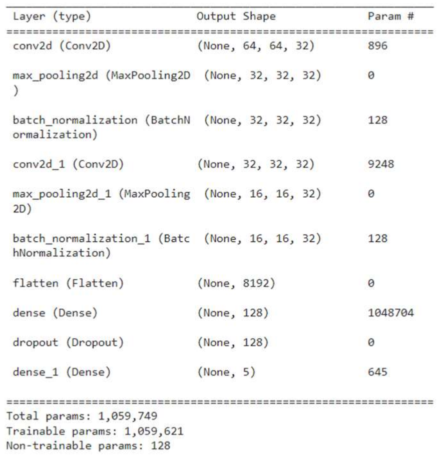
Then comes the second convolutional layer as a replica of the earlier layer for further feature extraction. Following this after finishing the previous two steps, we're supposed to have a pooled feature map by now. As the name of this step implies, we are literally going to flatten our pooled feature map into a column like matrix. There come the flatten module. This is then followed by a dense layer which is a layer that is deeply connected with its preceding layer which means the neurons of the layer are connected to every neuron of its preceding layer.

The dense layer's neuron in a model receives output from every neuron of its preceding layer, where neurons of the dense layer perform matrix-vector multiplication. A dropout layer was added after the densely connected layer to further prevent model from overfitting. The activation is 'softmax'. Softmax makes the output sum up to 1 so the output can be interpreted as probabilities. L1 regularizer was used in the dense layer. The model will then make its prediction based on which option has the highest probability.

To sum up, here is what we have after we're done with each of the steps that we have covered up until now:

Input image (starting point)

- Convolutional layer (convolution operation)
- Pooling layer (pooling)
- Input layer for the artificial neural network (flattening)
- Dense output layer with softmax activation function



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 64, 64, 32)        896

 max_pooling2d (MaxPooling2D  (None, 32, 32, 32)       0
 )

 batch_normalization (BatchN  (None, 32, 32, 32)       128
 ormalization)

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d_1 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 batch_normalization_1 (Batc  (None, 16, 16, 32)       128
 hNormalization)

 flatten (Flatten)           (None, 8192)              0

 dense (Dense)               (None, 128)               1048704

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 5)                 645

=================================================================
Total params: 1,059,749
Trainable params: 1,059,621
Non-trainable params: 128
```

Fig 3. Implemented CNN model

Impact of detailed image augmentations

Although variety of iterations were tried in the baseline 2 layer CNN model based on intuitions and various research papers the best one was retained and is being described in the report. The major impact was observed when detailed augmentations were carried out on the images during pre-processing this made a remarkable impact apart from adding regularizing parameters and tweaking the CNN filters. There was a drastic 0.1 increase in accuracy and significant impact on f1 score post augmentations. Prior to that even though the accuracy was acceptable enough the confusion matrix showed poor classification capabilities of the model. The previous model with no augmentations misclassified dot_line and line charts more_.

Also experimented with "squared hinge loss" and "categorical_cross entropy" loss function. The latter when combined with Adam optimizer provided quicker model convergence.

Comparative table of model accuracies and performance:

Model	**Additional modules**	**Accuracy/ Performance**
Custom 2 layer baseline CNN	Only pre processed input images	85%
Custom 2 layer baseline CNN	Pre processed input images with augmentations and square hinger loss function	95%
Custom 2 layer baseline CNN	Pre processed input images with augmentations and categorical_crossentropy	99% , quicker convergence

Further to aid efficient model training learning rate scheduler and model checkpoints function (in case of runtime crashes or large number of epochs when required during hyperprameter tuning) were utilized.

To conclude the created 2 layer CNN baseline model performed well with greater accuracy and lower convergence time upon iterative hyperparameter tuning of the model parameters and a detailed image augmentation process.

Moving on the use of pretrained models as applied to the above dataset will be discussed.

Approach 2 - Transfer learning

In this project, I will cover the top 4 pre-trained models for Image Classification that are state-of-the-art (SOTA) and are widely used in the industry as well and will be limiting the explanation to the current dataset in picture. Using a pretrained model is a highly effective approach, compared if you need to build it from scratch, where you need to collect great amounts of data and train it yourself.

<u>Inception</u>

Inception-ResNet-v2 is a convolutional neural network that is trained on more than a million images from the ImageNet database. The network is 164 layers deep and can classify images into 1000 object categories, such as the keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. For our dataset the input image size was modified to 128,128,3 from its usual 299x299 required image dimensions. Pretrained weights from the "imagenet" dataset were used. The convolutional layers were frozen to avoid retraining and facilitate way faster training time as compared to baseline model due to pretrained weights. The last layers of ANN were added as the dense layer for output of class predictions modified according to our dataset which has 5 classes. The pre-processing was done using the Keras provided preprocess_unit for inception_resnet_v2 and further image augmentations were performed. It was also observed that it is of utmost importance to use the corresponding preprocess_unit according to the model because without proper pre-processing the images are highly misclassified owing very high model loss during training even if a complex pretrained model is used.

<u>Xception</u>

Xception is a convolutional neural network that is 71 layers deep, it is also known as "extreme" version of an Inception module. The pre-processing used for this model was same as the one used for inception model, both performed poorly in it absence. For this model also the CNN layers were frozen and final dense layers were added with dropout for output prediction. It actually performed as well as inception but tad bit better in terms of accuracy with 0.001 difference and much quicker convergence than inception model. This converged quickest of all the models considered in this approach.

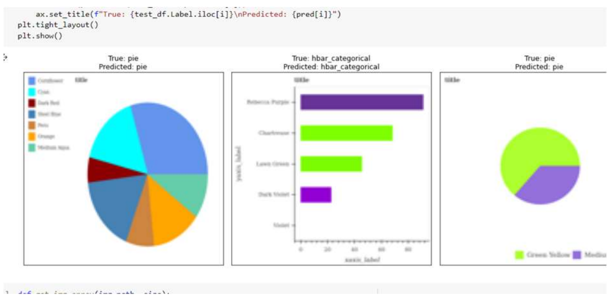
<u>Resnet50</u>

ResNet-50 is a pretrained Deep Learning model for image classification of the Convolutional Neural Network *(CNN, or ConvNet),* which is a class of deep neural networks, most commonly applied to analysing visual imagery. ResNet-50 is 50 layers deep and is trained on a million images of 1000 categories from the ImageNet database. Furthermore, the model has over 23 million trainable parameters, which indicates a deep architecture that makes it better for image recognition.

This model was used for classification task of out dataset by using transfer learning. The convolutional layers were frozen and on top of the pretrained resnet layers dense layers were added for modification according to the dataset concerned requirements. This model depicted the expected high level accuracy with quicker convergence as compared to the baseline model. The "imagenet" weights were used here.

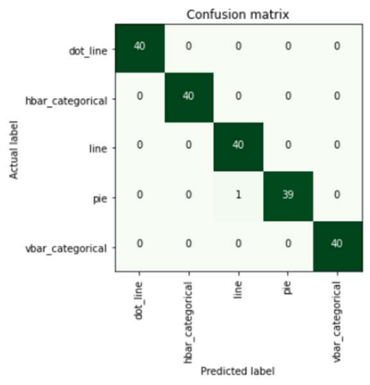
<u>VGG16</u>

VGG16 is a convolution neural net (CNN) architecture is considered to be one of the excellent vision model architectures till date. Most unique thing about VGG16 is that instead of having a large number of hyper-parameters they focused on having convolution layers of 3x3 filter with a stride 1 and always used same padding and maxpool layer of 2x2 filter of stride 2. It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture. In the end it has 2 FC(fully connected layers) followed by a softmax for output. The 16 in VGG16 refers to it has 16 layers that have weights. This network is a pretty large network and it has about 138 million (approx.) parameters. For the classification task at hand this pretrained model was used in a similar fashion to the Resnet model and performed equally well and took equally similar time as the Resnet model for attaining optimum accuracy while model training.



```
    ax.set_title(f"True: {test_df.Label.iloc[i]}\nPredicted: {pred[i]}")
plt.tight_layout()
plt.show()
```

Fig 4. Predictions obtained from VGG model on validation set



	dot_line	hbar_categorical	line	pie	vbar_categorical
dot_line	40	0	0	0	0
hbar_categorical	0	40	0	0	0
line	0	0	40	0	0
pie	0	0	1	39	0
vbar_categorical	0	0	0	0	40

Fig 5. The confusion matrix resembled more or less same like the one obtained for inception model

Comparative table of model accuracies and performance:

Model	**Additional modules**	**Accuracy and epochs**
Inception_ResnetV2	Without preprocess_unit (i.e without keeping range in between (0,1)	90% accuracy, very high loss. Poor model performance
Inception_ResnetV2	With preprocess_unit	99% accuracy, optimum accuracy at 10th epoch
Resnet50	With preprocess_unit (This was used for all pretrained models here on)	99% accuracy, optimum accuracy at 15th epoch
VGG16		99% accuracy, optimum accuracy at 3rd epoch
Xception		99% accuracy, optimum accuracy at 2nd epoch

All the pretrained model gave an excellent model performance on the validation sets and test sets, along with similar level custom test images. When it comes to making a choice for preferred model of the above used it will depend highly on system constraints, as in if the model is to be remotely deployed on a microcontroller we may prefer an smaller model with fewer parameters(for instance Squeezenet model could be tried), if accuracy was of extreme importance then any of the above model suits. Accordingly, the preference of the pretrained models changes based on constraints.

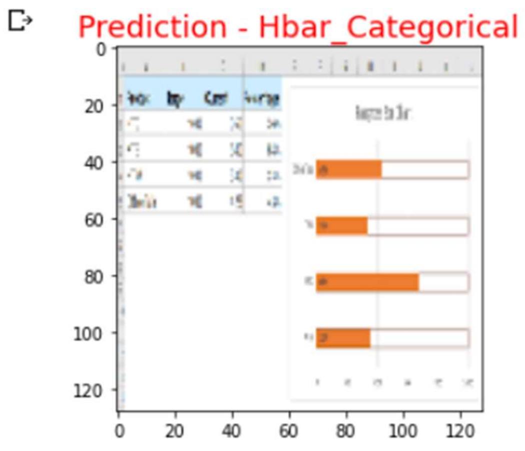


Fig 6. Prediction on **custom** test image (screen snip)

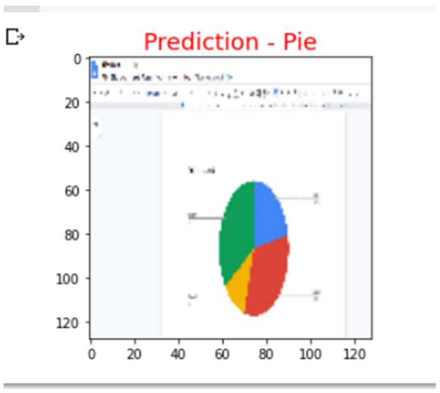


Fig 7. Predication on **custom** image

Use of GradCam

Taking the liberty f experimenting with the dataset, I proceeded with data augmentations which actually turned out to be a necessity of optimal performance on validation dataset. Further I tried my hand at the use of Gradcam for highlighting the object of concern the prediction of the input test image.

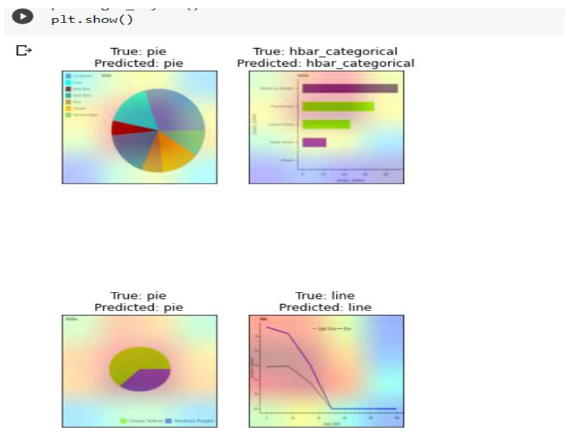
Gradient-weighted Class Activation Mapping (Grad-CAM), uses the gradients of any target concept (say 'dog' in a classification network or a sequence of words in captioning network) flowing into the final convolutional layer to produce a coarse localization map highlighting the important regions in the image for predicting the concept.

The technique is an improvement over previous approaches in versatility and accuracy. It is complex but, luckily, the output is intuitive. From a high-level, we take an image as input and create a model that is cut off at the layer for which we want to create a Grad-CAM heat-map. We attach the fully-connected layers for prediction. We then run the input through the model, grab the layer output, and loss. Next, we find the gradient of the output of our desired model layer w.r.t. the model loss.

If you go to look straight at the provided dataset and the results that were obtained using both approach 1 and 2 there would not really be a requirement for using Gradcam for the cause of increasing the classification ability, the reason it is being tried out here is to experiment and analyse the future scope of detecting charts (and highlighting its location in input) in documents or detecting multiple chart types in a single input (multiclass classification).

In the model I created above I used a pretrained model for training purpose. Post that I extracted the last convolutional layer and created GradCam heatmaps for the last convolutional layer. It gave satisfactory result on images within the scope of dataset.

This was also tried on custom images created using screen snipping and downloaded chart images from web. It performed well on images which only had the chart and a clear foreground and 2 out of 5 times performed well on images having multiple objects/charts. This is explainable and attributed to the fact that the training dataset did not have any such relevant images which were presented as a part of complex custom inputs. But it also implies that with further training dataset customization and required changes in model, the model along with GradCam will be able to detect charts from complex clustered test images and this can be tackled as a part of future scope for building further on current implementations.



```
plt.show()
```

Fig 8. GradCam output on dataset images

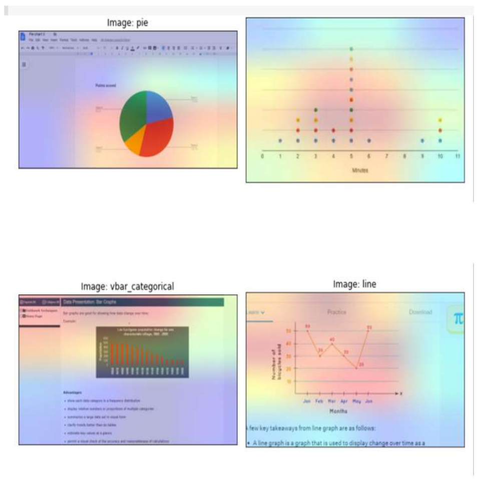


Fig 9. GradCam on **custom** created test images

Conclusion

The charts dataset classification problem statement was tackled using various approaches and all of them fared well with respect to classification on validation and test dataset. The key observations differentiating between these models were the time taken to attain the convergence and how data pre-processing and augmentations played a very crucial role before one could move in to hyperparameter tuning. Secondly the impact of using regularizing parameters was noted and was thus consistently used in all the models (for layers where required) build and utilized in the above task.