Genre Identification Utilizing Album Art

Glenn Connell
School of Computing
National College of Ireland
Dublin, Ireland
x14441832@student.ncirl.ie

Rohan Dongare
School of Computing
National College of Ireland
Dublin, Ireland
x18120199@student.ncirl.ie

Alfred Johnson
School of Computing
National College of Ireland
Dublin, Ireland
x17170141@student.ncirl.ie

Jay Shete
School of Computing
National College of Ireland
Dublin, Ireland
x18108458@student.ncirl.ie

***Abstract*— Contained within this document is a detailed, novel approach to the problem of genre classification. This approach attempts to produce a solution that can be utilized in place of the computationally intensive methods utilized within the industry currently. The approach utilizes a convolutional neural network in order to classify the musical genre of the album art input into the network.**

Keywords—Genre Classification, Genre Identification, Deep Neural Network, Artificial Neural Network, Convolutional Neural Network, DNN, CNN, ANN, Album Art Detection.

I. Introduction

Artificial Neural Networks (ANN) are a form of computation initially inspired by the function of the biological neural networks present in the brain. While they have since diverged from this emulation of biological functions it origin may be seen in the labelling of the components that comprise an ANN. ANNs typically consist of a collection of nodes or 'artificial neurons'. These neurons communicate data to one another via 'edges' and typically have an associated weight. This weight can be leveraged by the system to determine the value of the data possessed by a neuron and may be used to prune neurons that fail to contribute data that may overcome the weighting imposed upon it. Many implementations of ANNs utilize an aggregation of neurons to comprise several 'layers'. Each layer may be configured to perform specific operations upon the data, it is this adaptability that forms the foundations of the many variants of ANNs. The data flow of an ANN is typically feed-forward, starting with the data being entered into the input layer and then exiting via the output layer once it has passed through the varied hidden layers.

For decades ANNs have been the subject of intensive research and scrutiny, however this was hampered by the limitations of the technology available at the time. As technology advanced at an exponential rate these limitations have since been overcome, leading to prolific advancements in the field. ANNs have evolved to become a surgical tool rather than an all-encompassing approach to computing. A key branch of ANNs are Deep Neural Networks (DNN). DNN encompasses the popular evolutions of neural networks, Convolutional Neural Nets (CNN) and Recurrent Neural Nets (RNN). Both differ wildly in approach but are similar in effectiveness.

CNNs, similarly to ANNs, are inspired by the biological functions of the body. In this case the inspiration was drawn from the structure of the visual cortex with humans and the manner in which the brain processes imagery. CNNs incorporate several additional concepts into the traditional ANN formula. The defining feature of a CNN implementation is the inclusion of a convolutional layer. This layer is responsible for convolving the input data into a feature map, utilizing a 'kernel'. CNN implementations employ an alternating rotation of convolutional layers and pooling layers. Pooling layers are necessary to reduce dimensionality within the feature map and allows for the reduction of overfitting.

The motivation behind the undertaking of this project is the highly accessible nature of music. In recent years developments within data storage and music streaming have led directly to the rise of current digital media giants such as the subscription-based, Spotify and the popular audio distribution platform, Soundcloud. With the rise of these platforms comes the issue of scale. Initially, Soundcloud in particular, relied upon the userbase to classify the audio hosted upon the platform. However, while this approach was suited to the platform in its infancy, it was eventually replaced with complex algorithms, similar to the “Machine Listening” approach employed by Spotify. These algorithms observe audio at the molecular level in order to classify the music into its suited genre or sub-genre. This is an extremely successful approach to music classification however it comes with two large drawbacks: An in-depth knowledge of soundwaves alongside the structural composition of music and a large computation cost.

The business hypothesis for this project is the task of taking an input image, album-art for example, and outputting a probability of class that best describes its respective genre. The null hypothesis is to what extend an album-art could be used to classify a song's genre?

II. Related Work

Initial research conducted by [1] conceptualized a neural network model for pattern recognition that can be seen as a very early instance of a CNN. Inspired by the earlier work of [2], [3], [4], this research, while laying the groundwork for future research, is quite primitive when compared to more recent endeavors lacking many of the features commonly associated with neural networks. One of the most important of these missing features is the usage of the backpropagation algorithm [5], now a key component of modern implementations that enables a high degree of complexity to be reached in comparison to earlier works. A neural network conceptualized by [6] emulated many values of modern CNNs, including backpropagation, however it lacked key components such as pooling. Rather than convolving the features in a three dimensional space it this model was one-dimensional and simply performed upon a single axis, time. CNNs as we now recognize them where proposed by [7]. This paper examines the use of backpropagation on a neural network similar in structure to the neocognition network

outlined within the work of [1]. Also detailed within the paper are several variants of the proposed model, further refining the architecture.

[8] discusses the Regions with CNN features (RCNN) architecture. This architecture is an alteration to the CNN architecture in an attempt to create a scalable, accessible solution that boosts the mean average precision of the model. While RCNN achieves these goals it also encounters several issues, particularly in the training phase. [9] suggests an adaptation to the RCNN architecture, Fast RCNN. The Fast RCNN architecture is an adaptation of the RCNN architecture that aims to correct several issues present in the RCNN architecture. Fast RCNN aims to avoid the training difficulties inherent to the RCNN architecture, its multi-stage, time and spacial requirements.

Within the literature a variety of approaches to image and text recognition have been developed. The adaptation and improvisation of these techniques will be key to achieving a satisfactory outcome for this undertaking. The approaches drawn on for the purposes of this project are broad in scope, however, each provides a unique opportunity with distinct parameters that may provide an optimal solution for the addressed issue.

An SVM classification approach, as detailed in [10] and [11] have recently proven to perform to a desirable degree. These papers advocate an approach wherein the classifier considers both the image and the text within the image as features. This may serve as a key factor to the improvement of the recall of the classifications as many genres show an affinity to certain vocabulary.

A deep learning approach has been outlined within [12] and [13], however, this approach comes with difficulties that may impede the project such as the difficulty training the models necessary to conduct such an approach.

CNNs have been proven to perform exceptionally for a variety of image recognition and classification tasks, ranging from tasks such as the approach within [14] which addresses a similar problem we aim to address to tasks such as increasing the interpretability of CNNs such as the research proposed within [15]. Several other approaches such as the bag of words approach [16] and a decision tree approach [17] have also been proven to have merit within this field.

[18] describes a combination of the both CNN and RNN LS-TM architecture in an effort to achieve a solution that can make use of the benefits of both approaches. While this does indeed seem to increase performance of the model, the complexity of the approach is dramatically increased with a relatively low increase performance in comparison. So-called "C-LSTM" approaches tend to sacrifice complexity for the sake of a comparatively small increase in performance. While this can be valuable in certain scenarios, such as medical analyses, for the purposes of general NLP tasks this is likely overkill and a step many researchers will not wish to undertake

III. Data Mining Methodology

For the purposes of this project the Cross-Industry Process for Data Mining (CRISP-DM) methodology was adhered to. The CRISP-DM methodology consists of 6 stages. The first stage is the determination of business objectives. This is a crucial stage as it defines the goals of the project overall. For this project, as mentioned within Section I, the business objective is the creation of a novel approach to genre detection in an attempt to avoid the high computation costs of current approaches.

Stages 2 and 3 are closely intertwined. They consist of data understanding and data preparation respectively. For this project, these stages where handled simultaneously as the data utilized was created bespoke to the needs of the project. Stages 4, 5 and 6 consist of the modeling stage, evaluation stage and deployment stage. These stages contain the most intensive work pertaining to the machine learning aspects of the project. The details of these stages are contained within the remainder of this Section and Section IV.

A. *Input and Output*

The image data consists of images from 4 different genres, namely Metal, Rap, Trance, Country. Each genre is represented by 500 individual album-art. The resolution of each image 640 x 640 in JPEG form, which also determined the use of a 64 x 64 x 3 array of numbers to represent these images in the CNN. With color being a valuable aspect of identifying of each genre, the use of RGB values over greyscale has been taken into consideration. These factors compose arrays that represent the pixel intensity at every point using values from 0 to 255. These values are only the inputs available to the network when performing image classification. In theory a decent probability can be achieved indicating the genre of the album-art image using these arrays.

B. *Data Gathering*

The Data for the project was gathered using the publicly available Spotify API [1]. One of the most important considerations for the data was that all the images must have the same pixel density as well as aspect ratio. This was imperative as it would eliminate a lot of post processing. we managed to accomplish this objective in the data gathering process itself by making sure all the images taken had a size of 640 *x* 640 px and were of the same format i.e. JPEG. It was also made sure that there were no repeat albums downloaded. As this could negatively impact training. Each genre had a minimum of 500 images. The R programming language was used to create the code for this purpose as it has robust libraries for dealing with data cleaning and prep. All the code for the data gathering and cleansing process can be seen in the appendix.

C. *Workflow*

The overview of the CNN utilized for the purposes of this project is that the input, an image, is passed through a series of convolution layers, non-linear layers, pooling layers and fully connected layers in order to obtain an output that is then passed through the SoftMax activation function in order to receive a comprehensible result.

D. *Architecture*

The architecture of our network is shown in Fig 1. The CNN consists of three learned layers --- two convolutional

[1] https://developer.spotify.com/documentation/web-api/

and pooling layers then connected to one fully connected layer.

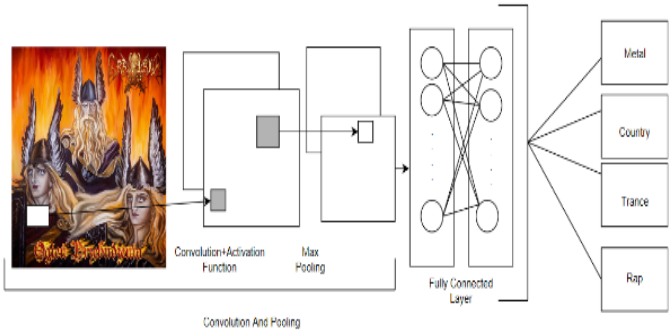

Figure 1: CNN Architecture

Neural Networks often run into training difficulties due to the saturation behavior of non-linear activation functions. Activation functions emulate the discrete switching of logic circuits which, in turn, deals with the non-linearity in the network. Non-linearity in a neural network can be both detrimental or beneficial, however. Beneficial as it allows the representation of more complicated functions and detrimental as this makes optimization more difficult. Rectified Linear Unit (ReLU) is the most common activation function, especially within CNNs. Mathematically, it is defined as y=max (0, X). Fig 2 contains a graphic representation of ReLU.

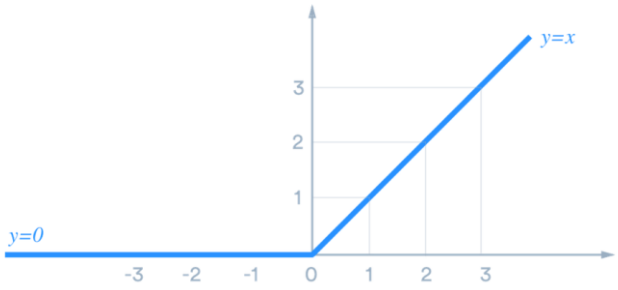

Figure 2: ReLU

Observable within Fig 2 is the tendency for ReLU to be linear for all positive values and zero for all negative values. Due to the simplified math this function is cheap to compute, and therefore, takes less time to train and run. The conversion for ReLU is faster as linearity brings slope and not a plateau to the gradient. Since ReLU is zero for all negative inputs its sparsely activated. All these reasons make this function computationally less expensive. For the purposes of this project, a large dataset is utilized thus, faster learning will have a great influence on model performance.

The nonlinear layer is followed by pooling layer, where a down-sampling operation is performed on the width and height of the album art. This reduces the volume of the image in order to avoid redundancy in the convolution process, as unimportant features may be removed via compression. In this research we have used the most common technique called Max-pooling.

The output of this layer is calculated by the maximum activation over non-overlapping regions of size (Px, Py), it creates position invariance over a larger local region and down-samples the image by factor of Px and Py along each dimension of a rectangular region of that image. This leads to a faster convergence rate by selecting superior features which improves the model performance with less computing power.

E. Handling the Problem of Overfitting

The neural network architecture has more than 2500 images to classify with four class labels with a 70/30 train/test split with various parameters in order to map album-art to its respective class. In practice the CNN turned out to be insufficient at learning this many parameters without considerable overfitting. Several measures have been undertaken to combat this overfitting problem.

The dataset consists of large image files and thus a large model that may take several hours to train and run. An efficient and computationally less expensive regularization technique called dropout has been utilized in this project. Dropout sets the output of each hidden neuron to zero with a probability of 0.2. Thus these 'dropped-out' neurons do not contribute to the forward pass and similarly do not participate in the back-propagation process for updating the weights. This technique reduces the complex dependency of any neuron in the architecture, which forces the network to learn more robust features that are more useful to avoid overfitting, along with a random sample subset passed to other neurons. However, during the test stage the outputs from all neurons are taken and multiplied by 0.2, which is a reasonable approximation of the dropout networks.

In the initial pass the model didn't know the values to choose for the weights and filter, as such they were randomized. The filter did not know which edges or curves to look for in a class. To resolve this backpropagation was undertaken. Backpropagation consists of 4 stages: the forward pass, the loss function, the backward pass and the weight update. Forward Pass: The training image is a 64 x 64 x 3 array of numbers with randomized weights and filters and as such the output fails to provide a conclusion as to the class of the image. Loss Function: A loss function consists of a Mean Squared Error (MSE), which is .5 times the actual-predicted squared error. The loss was extremely high for first couple of training images, to combat this, the loss function was minimized.

$$E_{Total} = \sum \frac{1}{2} (\text{target} - \text{output})^2$$

Once this has occurred a backward pass through the network is undertaken. This determines which weights correspond the most to the loss function, thus finding ways to reduce this loss. The below equation gives us this value by taking the derivative which, can then be used to update the weights.

$$w = w_i - \acute{\eta} \frac{dL}{dw}$$

Where: w = weight
w_i = initial weight
$\acute{\eta}$ = learning rate

The learning rate is a hyperparameter which is used to set the rate at which convergence of an optimal weight is set. In this project we have set this value to .01 as too high a value could lead to an imprecise value for optimal weights.

F. *Hyper-Parameter Optimisation*

As running hyper parameter optimization was computationally not feasible for the entire dataset, a batch of 500 samples was taken during every iteration of a random search. Nadam, with 5 epochs, a dropout 0.2 and a batch-size of 40 gives the highest accuracy of 75.25%. The model with 5 epochs for the entire dataset gave an accuracy of 72%.

Optimizer	Epochs	Dropout	Batch Size	Accuracy (in %)
Adamax	10	0.8	20	75.00
Adamax	15	0.2	40	73.70
Adamax	15	0.4	20	73.75
Nadam	**5**	**0.2**	**40**	**75.25**
Adam	5	0.0	20	73.7
SGD	10	0.4	40	75
Nadam	10	0.8	20	73.5
Adam	5	0.4	30	75
SGD	15	0.4	30	73.2
Nadam	10	0.4	30	69.75

Figure 3 Random Search Optimsation output

Grid search was also implemented for the project to fine tune the hyper-parameters and the output is added in the appendix. Grid search gave the best accuracy for the 'Nadam' Optimizer with drop-out set to 0.2, 10 epochs and batch size of 30. Grid search also returned the same hyperparameters as random search except the number of epochs was 10 as opposed to 5 suggested by random search. The model reported an accuracy of 75.38 for 5 epochs and 77.51 for 10 epochs for the entire dataset, while the other parameters were kept the same as reported by the output of grid and random search.

IV. Evaluation

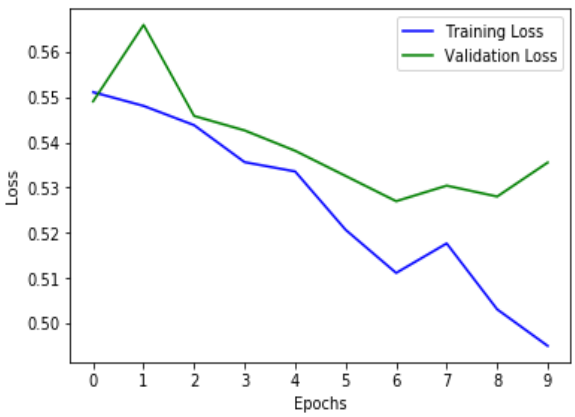

Figure 4 Training vs Validation Loss

The accuracy for the validation set reported by the model was 75.19%, and the validation loss reported was 0.55. The training and validation loss during the initial epochs did not report a large difference, which indicates that the model has high bias and low variance. As the number of epochs increases, the difference between the validation loss and training loss keeps increasing. Although after 8 epochs the difference increases which suggests that the model is overfitting the data.

```
              precision    recall  f1-score   support

     Country       0.33      0.54      0.41       181
       Metal       0.07      0.02      0.03       103
         Rap       0.25      0.31      0.28       126
      Trance       0.16      0.06      0.09       110

   micro avg       0.28      0.28      0.28       520
   macro avg       0.20      0.23      0.20       520
weighted avg       0.22      0.28      0.23       520

[[97  9 58 17]
 [66  2 28  7]
 [66  9 39 12]
 [65  8 30  7]]
```

Figure 5 Confusion matrix

Country and Rap genres reported high precision of 33% and 25% respectively, and also reported a high recall of 54% and 31% respectively.

A precision of 33% for country music indicates that out of all the country music album arts present in the dataset the model accurately classified 33% of them to be as country music album arts. Recall is a measure of completeness and a recall of 54% for the country class indicates that out of all the album arts present in the dataset, the model correctly classified 54% of the country music album arts. Out of the four classes metal and trance showed a very low precision and recall and efforts were made to improve the model performance.

One technique of improving the model performance was saving the best weights using model checkpoints and then while executing the model again these weights were reused and if the accuracy of the model improved the new weights were stored again and reused in the future iterations as the base or starting weights. So instead of starting with random weights we start with weights that have proven to be give better results.

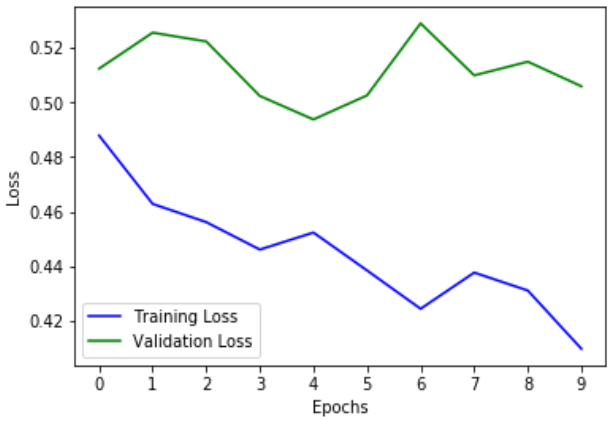

Figure 6: Training vs Validation with best weights

With 10 epochs and best weights from the previous iteration, both the training and validation loss decrease but the training loss decreases substantially, hinting overfitting of the data. The validation accuracy increased to 77% and the loss also decreased to 0.51 compared to the initial model.

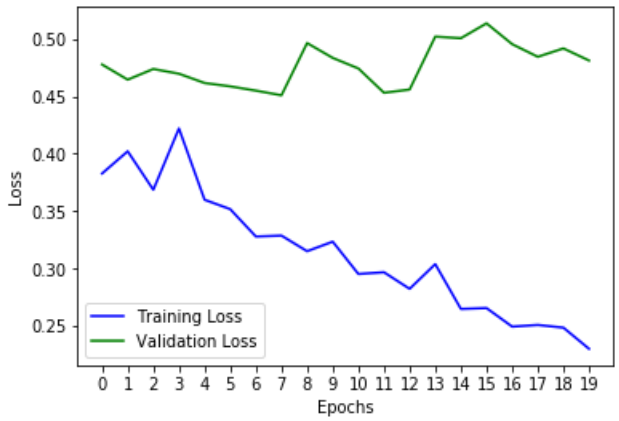

Figure 7 Training Vs Validation For 20 Epochs and Best Weights

With 20 epochs and best weights the validation accuracy jumped to 81.73% and the validation loss decreased to 0.48 while the margin between training and validation loss increased drastically. So, it is evident that with increase in epochs the model will overfit but a possible solution to this would be increasing the dropout.

	precision	recall	f1-score	support
Country	0.31	0.34	0.33	181
Metal	0.23	0.34	0.28	103
Rap	0.24	0.17	0.20	126
Trance	0.22	0.17	0.19	110
micro avg	0.26	0.26	0.26	520
macro avg	0.25	0.26	0.25	520
weighted avg	0.26	0.26	0.26	520

```
[[62 55 34 30]
 [33 35 18 17]
 [53 32 21 20]
 [49 27 15 19]]
```

The confusion matrix for the model with best weights and 20 epochs does not show a lot of improvement in precision values for country and rap classes which were already high but shows a significant improvement for Metal and Trance class compared to our initial model.

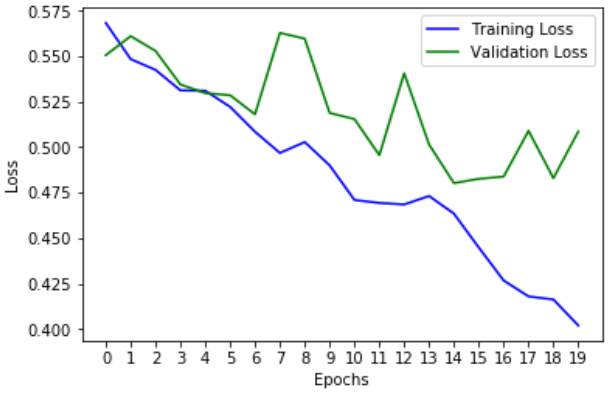

Figure 8 Training Vs Validation Loss with increased drop-out

With the same parameters as before (20 epochs and best weights) the drop out was increased to 0.4 as opposed to the original 0.2. The model accuracy was now 77.12% with a validation loss of 0.50. The difference in validation and training loss also decreased as the dropout was increased, which meant that despite the reduction in accuracy the model was now more generalized i.e. would perform better for unseen data.

Conclusions And Future Work

In conclusion, within this paper a novel approach to genre identification utilizing album-art is detailed. This approach avoids the heavy computational requirements that current approaches employed by commercial organizations. While not boasting an extreme measure of efficiency, this can be rectified with further optimizations to the architecture of the CNN employed.

Regarding future work based upon this work, the implementation of key RNN components into the structure of the CNN utilized for this project could prove to be key to improving classification capabilities. Various amalgamation approaches have been proven to be more successful at certain tasks when compared to more traditional approaches within the literature, however, this often comes with a hefty computation cost. If this could be circumvented it could prove key to success within this area.

References

[1] Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for amechanism of pattern recognition unaffected by shift in position,Biological cybernetics36(4): 193–202.

[2] Hubel, D. H. and Wiesel, T. N. (1962). Receptive fields, binocular interaction and func-tional architecture in the cat's visual cortex,The Journal of physiology160(1): 106–154.

[3] Hubel, D. H. and Wiesel, T. N. (1968). Receptive fields and functional architecture ofmonkey striate cortex,The Journal of physiology195(1): 215–243.

[4] Hubel, D. H. and Wiesel, T. N. (1977). Ferrier lecture: Functional architecture of macaquemonkey visual cortex,Proceedings of the Royal Society of London. Series B, BiologicalSciencespp. 1–59.

[5] Hecht-Nielsen, R. (1992). Theory of the backpropagation neural network,Neural networksfor perception, Elsevier, pp. 65–93.

[6] Waibel, A., Hanazawa, T., Hinton, G., Shikano, K. and Lang, K. J. (1995). Phonemerecognition using time-delay neural networks,Backpropagation: Theory, Architecturesand Applicationspp. 35–61.

[7] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. andJackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition,Neural computation1(4): 541–551.

[8] Girshick, R., Donahue, J., Darrell, T. and Malik, J. (2014). Rich feature hierarchiesfor accurate object detection and semantic segmentation,Proceedings of the IEEEconference on computer vision and pattern recognition, pp. 580–587.

[9] Girshick, R. (2015). Fast r-cnn,Proceedings of the IEEE international conference oncomputer vision, pp. 1440–1448.

[10] L. Guo-Shuai , Y. Fei , . L. Zhen-Bo and L. Cheng-Lin, "Fast Genre Classification of Web Images Using Global and Local Features," in 2017 4th IAPR Asian Conference on Pattern Recognition (ACPR), Nanjing, China, 2017.

[11] V. V. Palkar and P. Joeg, "Proposing scalable method for music genre classification," 2016 International Conference on Inventive Computation Technologies (ICICT), Coimbatore, 2016, pp. 1-6.

[12] He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[13] Chan, T.H., Jia, K., Gao, S., Lu, J., Zeng, Z. and Ma, Y., 2015. PCANet: A simple deep learning baseline for image classification?. IEEE Transactions on Image Processing, 24(12), pp.5017-5032.

[14] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep con-volutional neural networks. In: Advances in neural information processing systems.pp. 1097–1105 (2012).

[15] Zhang, Q., Nian Wu, Y. and Zhu, S.C., 2018. Interpretable convolutional neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 8827-8836).

[16] Z. Xiangrong , J. Kai , . Z. Yaoguo, A. Jinliang , H. Yanning and . J. Licheng, "Spatially constrained Bag-of-Visual-Words for hyperspectral image classification," 2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS), 2016.

[17] R. Hicham , E. Abdelmajid , B.-H. Abderrahim and B. Farid , "Classification and Recognition of Dental Images Using a Decisional Tree," 2016 13th International Conference on Computer Graphics, Imaging and Visualization (CGiV), 2016.

[18] Zhou, C., Sun, C., Liu, Z. and Lau, F. (2015). A c-lstm neural network for text classific-ation,arXiv preprint arXiv:1511.08630.

V. APPENDIX

Output Of Grid Search:

```
Best Reported Accuracy: 0.755000 using parameters {'batch_size': 30, 'dropout_rate': 0.2, 'epochs': 10, 'optimizer': 'Nadam'}
0.750000 (0.000000) with: {'batch_size': 20, 'dropout_rate': 0.0, 'epochs': 5, 'optimizer': 'SGD'}
0.745000 (0.006966) with: {'batch_size': 20, 'dropout_rate': 0.0, 'epochs': 5, 'optimizer': 'Adam'}
0.750000 (0.000000) with: {'batch_size': 20, 'dropout_rate': 0.0, 'epochs': 5, 'optimizer': 'Adamax'}
0.732500 (0.036105) with: {'batch_size': 20, 'dropout_rate': 0.0, 'epochs': 5, 'optimizer': 'Nadam'}
0.750000 (0.000000) with: {'batch_size': 20, 'dropout_rate': 0.0, 'epochs': 10, 'optimizer': 'SGD'}
0.712500 (0.034474) with: {'batch_size': 20, 'dropout_rate': 0.0, 'epochs': 10, 'optimizer': 'Adam'}
0.727500 (0.026912) with: {'batch_size': 20, 'dropout_rate': 0.0, 'epochs': 10, 'optimizer': 'Adamax'}
0.722500 (0.043259) with: {'batch_size': 20, 'dropout_rate': 0.0, 'epochs': 10, 'optimizer': 'Nadam'}
0.750000 (0.000000) with: {'batch_size': 20, 'dropout_rate': 0.2, 'epochs': 5, 'optimizer': 'SGD'}
0.747500 (0.003483) with: {'batch_size': 20, 'dropout_rate': 0.2, 'epochs': 5, 'optimizer': 'Adam'}
0.750000 (0.000000) with: {'batch_size': 20, 'dropout_rate': 0.2, 'epochs': 5, 'optimizer': 'Adamax'}
0.752500 (0.003562) with: {'batch_size': 20, 'dropout_rate': 0.2, 'epochs': 5, 'optimizer': 'Nadam'}
0.750000 (0.000000) with: {'batch_size': 20, 'dropout_rate': 0.2, 'epochs': 10, 'optimizer': 'SGD'}
0.742500 (0.000106) with: {'batch_size': 20, 'dropout_rate': 0.2, 'epochs': 10, 'optimizer': 'Adam'}
0.737500 (0.022901) with: {'batch_size': 20, 'dropout_rate': 0.2, 'epochs': 10, 'optimizer': 'Adamax'}
0.700000 (0.008831) with: {'batch_size': 20, 'dropout_rate': 0.2, 'epochs': 10, 'optimizer': 'Nadam'}
0.750000 (0.000000) with: {'batch_size': 20, 'dropout_rate': 0.4, 'epochs': 5, 'optimizer': 'SGD'}
0.750000 (0.000000) with: {'batch_size': 20, 'dropout_rate': 0.4, 'epochs': 5, 'optimizer': 'Adam'}
0.750000 (0.000000) with: {'batch_size': 20, 'dropout_rate': 0.4, 'epochs': 5, 'optimizer': 'Adamax'}
0.725000 (0.035622) with: {'batch_size': 20, 'dropout_rate': 0.4, 'epochs': 5, 'optimizer': 'Nadam'}
0.750000 (0.000000) with: {'batch_size': 20, 'dropout_rate': 0.4, 'epochs': 10, 'optimizer': 'SGD'}
0.745000 (0.024778) with: {'batch_size': 20, 'dropout_rate': 0.4, 'epochs': 10, 'optimizer': 'Adam'}
0.727500 (0.024711) with: {'batch_size': 20, 'dropout_rate': 0.4, 'epochs': 10, 'optimizer': 'Adamax'}
0.705000 (0.021382) with: {'batch_size': 20, 'dropout_rate': 0.4, 'epochs': 10, 'optimizer': 'Nadam'}
0.750000 (0.000000) with: {'batch_size': 20, 'dropout_rate': 0.6, 'epochs': 5, 'optimizer': 'SGD'}
0.747500 (0.003483) with: {'batch_size': 20, 'dropout_rate': 0.6, 'epochs': 5, 'optimizer': 'Adam'}
0.750000 (0.000000) with: {'batch_size': 20, 'dropout_rate': 0.6, 'epochs': 5, 'optimizer': 'Adamax'}
0.722500 (0.038315) with: {'batch_size': 20, 'dropout_rate': 0.6, 'epochs': 5, 'optimizer': 'Nadam'}
0.750000 (0.000000) with: {'batch_size': 20, 'dropout_rate': 0.6, 'epochs': 10, 'optimizer': 'SGD'}
0.750000 (0.000000) with: {'batch_size': 20, 'dropout_rate': 0.6, 'epochs': 10, 'optimizer': 'Adam'}
0.737500 (0.006861) with: {'batch_size': 20, 'dropout_rate': 0.6, 'epochs': 10, 'optimizer': 'Adamax'}
0.702500 (0.028863) with: {'batch_size': 20, 'dropout_rate': 0.6, 'epochs': 10, 'optimizer': 'Nadam'}
0.750000 (0.000000) with: {'batch_size': 20, 'dropout_rate': 0.8, 'epochs': 5, 'optimizer': 'SGD'}
0.750000 (0.000000) with: {'batch_size': 20, 'dropout_rate': 0.8, 'epochs': 5, 'optimizer': 'Adam'}
0.750000 (0.000000) with: {'batch_size': 20, 'dropout_rate': 0.8, 'epochs': 5, 'optimizer': 'Adamax'}
0.750000 (0.000000) with: {'batch_size': 20, 'dropout_rate': 0.8, 'epochs': 5, 'optimizer': 'Nadam'}
0.750000 (0.000000) with: {'batch_size': 20, 'dropout_rate': 0.8, 'epochs': 10, 'optimizer': 'SGD'}
0.750000 (0.000000) with: {'batch_size': 20, 'dropout_rate': 0.8, 'epochs': 10, 'optimizer': 'Adam'}
0.750000 (0.000000) with: {'batch_size': 20, 'dropout_rate': 0.8, 'epochs': 10, 'optimizer': 'Adamax'}
0.732500 (0.012737) with: {'batch_size': 20, 'dropout_rate': 0.8, 'epochs': 10, 'optimizer': 'Nadam'}
0.750000 (0.000000) with: {'batch_size': 30, 'dropout_rate': 0.0, 'epochs': 5, 'optimizer': 'SGD'}
0.725000 (0.025703) with: {'batch_size': 30, 'dropout_rate': 0.0, 'epochs': 5, 'optimizer': 'Adam'}
```

```
0.750000 (0.000000) with: {'batch_size': 20, 'dropout_rate': 0.8, 'epochs': 5, 'optimizer': 'Nadam'}
0.750000 (0.000000) with: {'batch_size': 20, 'dropout_rate': 0.8, 'epochs': 10, 'optimizer': 'SGD'}
0.750000 (0.000000) with: {'batch_size': 20, 'dropout_rate': 0.8, 'epochs': 10, 'optimizer': 'Adam'}
0.750000 (0.000000) with: {'batch_size': 20, 'dropout_rate': 0.8, 'epochs': 10, 'optimizer': 'Adamax'}
0.732500 (0.012737) with: {'batch_size': 20, 'dropout_rate': 0.8, 'epochs': 10, 'optimizer': 'Nadam'}
0.750000 (0.000000) with: {'batch_size': 30, 'dropout_rate': 0.0, 'epochs': 5, 'optimizer': 'SGD'}
0.725000 (0.025703) with: {'batch_size': 30, 'dropout_rate': 0.0, 'epochs': 5, 'optimizer': 'Adam'}
0.742500 (0.010687) with: {'batch_size': 30, 'dropout_rate': 0.0, 'epochs': 5, 'optimizer': 'Adamax'}
0.715000 (0.025165) with: {'batch_size': 30, 'dropout_rate': 0.0, 'epochs': 5, 'optimizer': 'Nadam'}
0.750000 (0.000000) with: {'batch_size': 30, 'dropout_rate': 0.0, 'epochs': 10, 'optimizer': 'SGD'}
0.722500 (0.019625) with: {'batch_size': 30, 'dropout_rate': 0.0, 'epochs': 10, 'optimizer': 'Adam'}
0.725000 (0.017551) with: {'batch_size': 30, 'dropout_rate': 0.0, 'epochs': 10, 'optimizer': 'Adamax'}
0.732500 (0.019862) with: {'batch_size': 30, 'dropout_rate': 0.0, 'epochs': 10, 'optimizer': 'Nadam'}
0.750000 (0.000000) with: {'batch_size': 30, 'dropout_rate': 0.2, 'epochs': 5, 'optimizer': 'SGD'}
0.750000 (0.000000) with: {'batch_size': 30, 'dropout_rate': 0.2, 'epochs': 5, 'optimizer': 'Adam'}
0.742500 (0.016284) with: {'batch_size': 30, 'dropout_rate': 0.2, 'epochs': 5, 'optimizer': 'Adamax'}
0.695000 (0.078369) with: {'batch_size': 30, 'dropout_rate': 0.2, 'epochs': 5, 'optimizer': 'Nadam'}
0.750000 (0.000000) with: {'batch_size': 30, 'dropout_rate': 0.2, 'epochs': 10, 'optimizer': 'SGD'}
0.732500 (0.040847) with: {'batch_size': 30, 'dropout_rate': 0.2, 'epochs': 10, 'optimizer': 'Adam'}
0.740000 (0.007177) with: {'batch_size': 30, 'dropout_rate': 0.2, 'epochs': 10, 'optimizer': 'Adamax'}
0.755000 (0.017758) with: {'batch_size': 30, 'dropout_rate': 0.2, 'epochs': 10, 'optimizer': 'Nadam'}
0.750000 (0.000000) with: {'batch_size': 30, 'dropout_rate': 0.4, 'epochs': 5, 'optimizer': 'SGD'}
0.750000 (0.000000) with: {'batch_size': 30, 'dropout_rate': 0.4, 'epochs': 5, 'optimizer': 'Adam'}
0.745000 (0.007124) with: {'batch_size': 30, 'dropout_rate': 0.4, 'epochs': 5, 'optimizer': 'Adamax'}
0.720000 (0.041798) with: {'batch_size': 30, 'dropout_rate': 0.4, 'epochs': 5, 'optimizer': 'Nadam'}
0.750000 (0.000000) with: {'batch_size': 30, 'dropout_rate': 0.4, 'epochs': 10, 'optimizer': 'SGD'}
0.747500 (0.003562) with: {'batch_size': 30, 'dropout_rate': 0.4, 'epochs': 10, 'optimizer': 'Adam'}
0.750000 (0.006109) with: {'batch_size': 30, 'dropout_rate': 0.4, 'epochs': 10, 'optimizer': 'Adamax'}
0.725000 (0.024821) with: {'batch_size': 30, 'dropout_rate': 0.4, 'epochs': 10, 'optimizer': 'Nadam'}
0.750000 (0.000000) with: {'batch_size': 30, 'dropout_rate': 0.6, 'epochs': 5, 'optimizer': 'SGD'}
0.747500 (0.003483) with: {'batch_size': 30, 'dropout_rate': 0.6, 'epochs': 5, 'optimizer': 'Adam'}
0.747500 (0.003562) with: {'batch_size': 30, 'dropout_rate': 0.6, 'epochs': 5, 'optimizer': 'Adamax'}
0.722500 (0.019625) with: {'batch_size': 30, 'dropout_rate': 0.6, 'epochs': 5, 'optimizer': 'Nadam'}
0.750000 (0.000000) with: {'batch_size': 30, 'dropout_rate': 0.6, 'epochs': 10, 'optimizer': 'SGD'}
0.745000 (0.003510) with: {'batch_size': 30, 'dropout_rate': 0.6, 'epochs': 10, 'optimizer': 'Adam'}
0.747500 (0.003483) with: {'batch_size': 30, 'dropout_rate': 0.6, 'epochs': 10, 'optimizer': 'Adamax'}
0.727500 (0.018602) with: {'batch_size': 30, 'dropout_rate': 0.6, 'epochs': 10, 'optimizer': 'Nadam'}
0.750000 (0.000000) with: {'batch_size': 30, 'dropout_rate': 0.8, 'epochs': 5, 'optimizer': 'SGD'}
0.750000 (0.000000) with: {'batch_size': 30, 'dropout_rate': 0.8, 'epochs': 5, 'optimizer': 'Adam'}
0.747500 (0.003562) with: {'batch_size': 30, 'dropout_rate': 0.8, 'epochs': 5, 'optimizer': 'Adamax'}
0.730000 (0.027865) with: {'batch_size': 30, 'dropout_rate': 0.8, 'epochs': 5, 'optimizer': 'Nadam'}
0.750000 (0.000000) with: {'batch_size': 30, 'dropout_rate': 0.8, 'epochs': 10, 'optimizer': 'SGD'}
0.750000 (0.000000) with: {'batch_size': 30, 'dropout_rate': 0.8, 'epochs': 10, 'optimizer': 'Adam'}
0.747500 (0.003562) with: {'batch_size': 30, 'dropout_rate': 0.8, 'epochs': 10, 'optimizer': 'Adamax'}
```