CNN Assignment

The aim of assignment 2 was to train a convolutional neural network to be able to predict the tags of steam videogames from screenshots of the game. The dataset used consisted of a total of 128,576 screenshots from 14,687 different videogames, with each videogame having one or more of 441 tags, with tags consisting of things such as ‘FPS’, ‘Family-friendly’, ‘Indie’. Thus, the problem at hand was that of a multilabel classification nature. As the degree of classification was extremely large, and the content and styling of the images within the dataset was of a very diverse nature (see images below), a complex model would be needed to achieve a high accuracy score. Because of this, transfer learning would be employed from a pretrained model, with layers on top to tailor the model to our specific dataset. Keras and Tensorflow were the main libraries used to build the network.

Data Preprocessing

As previously stated, there was a total of 441 possible tags a videogame could be labelled with. The descriptor of these tags varied widely, from videogame genres, to game aesthetics and also descriptors that had no relation to the game’s content at all. As well as this, the popularity of the tags varied widely, with many tags only used for a select amount of games. As such, the model would be incapable of learning to correctly label games according to all of these tags, and a filtering process was necessary in order to identify which tags were to be chosen from. Initially, this process consisted of first identifying and removing the tags which were deemed to have little to no association with the content and gameplay of the game, and thus would not be able to be predicted based on screenshots (e.g, ‘Indie’, ‘Early Access’, ‘Multiple Endings’). Tags that were deemed to be no more than the combination of two other tags (‘Dark Fantasy’), or where the meaning of the tag was ambiguous (‘Education), were also removed, Next, any tags which were used in less than 100 videogames were removed (158 tags). However, even after this filtering, many tags remained which the model was failing to learn. As said before, this was likely due to the complexity and variety of images in the dataset- the types of games, content and graphics could all be completely different from videogame to videogame. It was also likely due to the fact that many tags were not represented in every (or even any) screenshots of the games (e.g ‘Shooter’ tag with no guns present in a menu or dialogue screenshot, or ‘Sexual Content’ with sexual content only present in one of many videogame screenshots). For these reasons, the model needed many more instances in order to correctly identify the aspects of the game correctly. The minimum number of tag examples was increased from 100 to 500 based on previous model performance, and outside of these criteria, a select number of tags were also removed which the model repeatedly failed to learn to correctly identify. After this filtering process, the final

list of labels the model was to aim to predict consisted of 28 tags, and can be found in the table below.

After finalizing the list of tags the model was going to aim to predict, the data was ready to be split. To avoid data leakage, the data was split by videogame as opposed to screenshots. There were two options for conducting the data split- firstly cross validation could be employed to train, validate and choose the best model before a final holdout set measured its accuracy, or secondly a train-test-validation split could be conducted, where the validation set was to be used to test the models being trained, and the test set was used as a final holdout set to measure the performance of the final model picked. The latter option was chosen here for the reason of computational efficiency, a large factor to be considered in the current analysis. In order to conduct the train-test-validation split in a way where each tag was represented proportionally evenly in each dataset, a multilabel iterative stratification algorithm was used. This ensured each tag was well-represented in each dataset. The split used was an 80-10-10, as the amount of data used to train the model needed to be maximized for reasons already stated, while it was important that the validation and test metrics were relatively reliable and consistent.

Image Preprocessing

The next step was to preprocess the images in the dataset. Most images in the original dataset had a resolution of 1920x1080, with some proportion of them having varying resolutions smaller than this. The target size resolution that needed to be inputted into the model was 299x299. Therefore the images needed to be shrunk dramatically. In order to maintain the aspect ratio and not decrease the resolution of the inputted images by too great a degree, we initially randomly cropped each image first to a size of 897x897, and then shrunk these randomly cropped images by a factor of 3 to the target size. However, it was later revealed that this method brought about significantly worse results than simply resizing the original images to the target size. Thus, no cropping was used. Regarding data augmentation, a number of techniques were initially employed, such as random shearing, recolouring and flips. However, these methods again did not bring about any increases in performance, and for this reason were not used in the training of the final model.

Building the Model

As previously stated, due to the complexity of the dataset and the classification problem the model would be employed to complete, as well as computational limitations, transfer learning was employed to finetune a pre-trained model to our problem. The Xception model with weights trained on the ImageNet dataset was employed, as this is a very efficient model that has been previously shown to work well on similar datasets. The top layer of the model was removed, and all other layers were frozen so that no training would take place on them. On top of the Xception layers, a Flatten layer was placed, followed by four dense layers. Batch normalization and dropout was also employed after each dense layer. The model's output layer consisted of 28 neurons with sigmoid activations to treat output as independent. There

were a number of hyperparameters that still needed to be chosen from here- the number of neurons in each dense layer, the optimizer to be used and its initial learning rate, the dropout rate, the activation functions to be used and also the loss function. To choose all of these hyperparameters, a random search using Keras Tuner was employed to train models with random combinations of these hyperparameters and return the best performing one. A full summary of all of the parameter values tested is found in the table below.

Training and Choosing the Model

Due to memory constraints, all of the images in the training dataset could not be loaded into memory at once, and so an image data generator pipeline was used to load images into memory and train the model on them in batches. A batch size of 32 was used for training, meaning tensors of size 32x299x299x3 were passed through the model. For each combination of model hyperparameters tested, the model was trained for three epochs (more epochs here were not possible due to computational limitations), and each combination was trained twice. The models were evaluated based on their validation F1 scores (not on validation loss as different loss functions were being employed). The best performing model hyperparameters were deemed to be dense layers with 512, 256, 256 and 128 neurons respectively. The optimizer chosen was the Adam optimizer with an initial learning rate close to .0001. The regular ReLu activation function was used. The chosen dropout rate was .25 and the loss function was binary cross entropy. Thus, this model was chosen as the final model. The model was next furtherly trained until early stopping revealed no further improvement to the validation loss. This led to the model being trained for a total of 5 epochs, with the final validation loss being .38

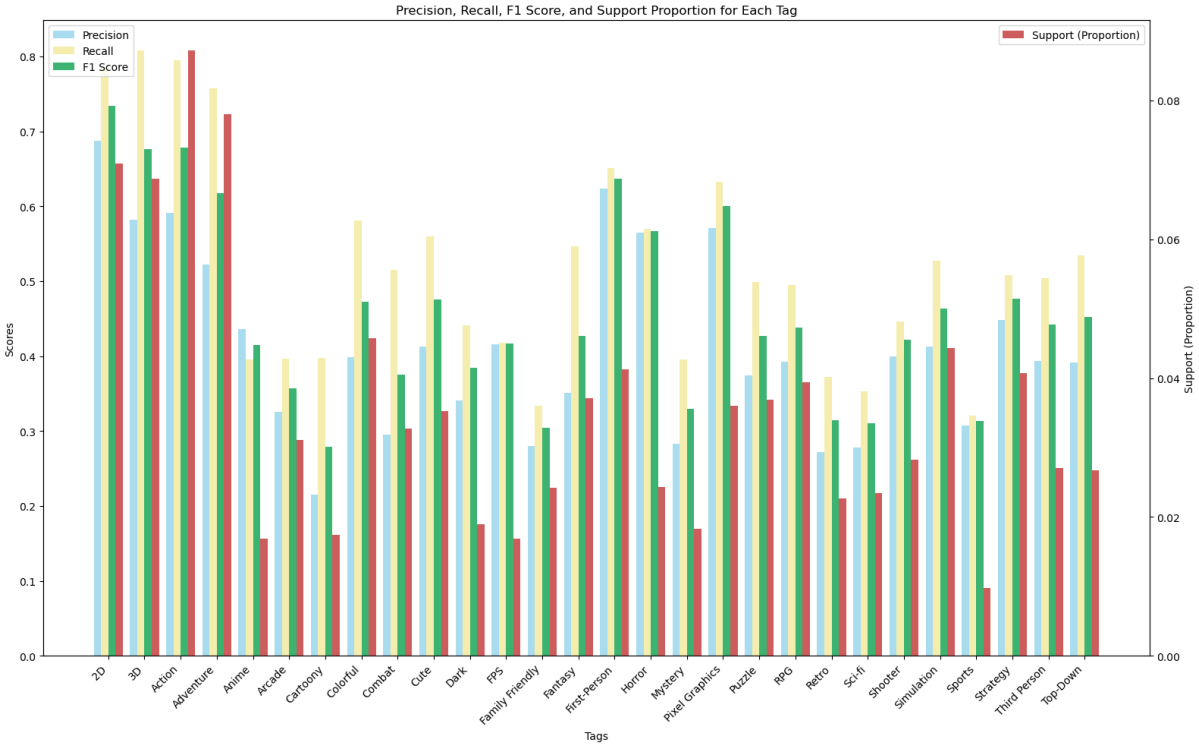
Customizing Classification Thresholds for Each Tag

The three main metrics used to interpret the accuracy of the model were recall, precision and F1 scores. Recall is the fraction of true labels that are correctly identified by the model, while precision is the fraction of predicted labels that are actually true. The F1 score is the harmonic mean of precision and recall. At this point, the model had achieved a recall score of .57, and precision score of .43, and an F1 score of .49 using a cutoff threshold of .25 for classifying all tags. However, the optimal threshold for maximizing correct classification and thus F1 scores will differ from tag to tag, largely depending on how well represented the tag was in the dataset. Thus, in order to furtherly increase the accuracy of the model, the optimal threshold where the F1 score was maximized based on training data predictions was found for each tag and was then used to predict the tags in the validation and test data after. The custom thresholds led to a validation recall score of .58, precision score of .46 and an F1 score of .51, thus a slight increase in model accuracy.

Results

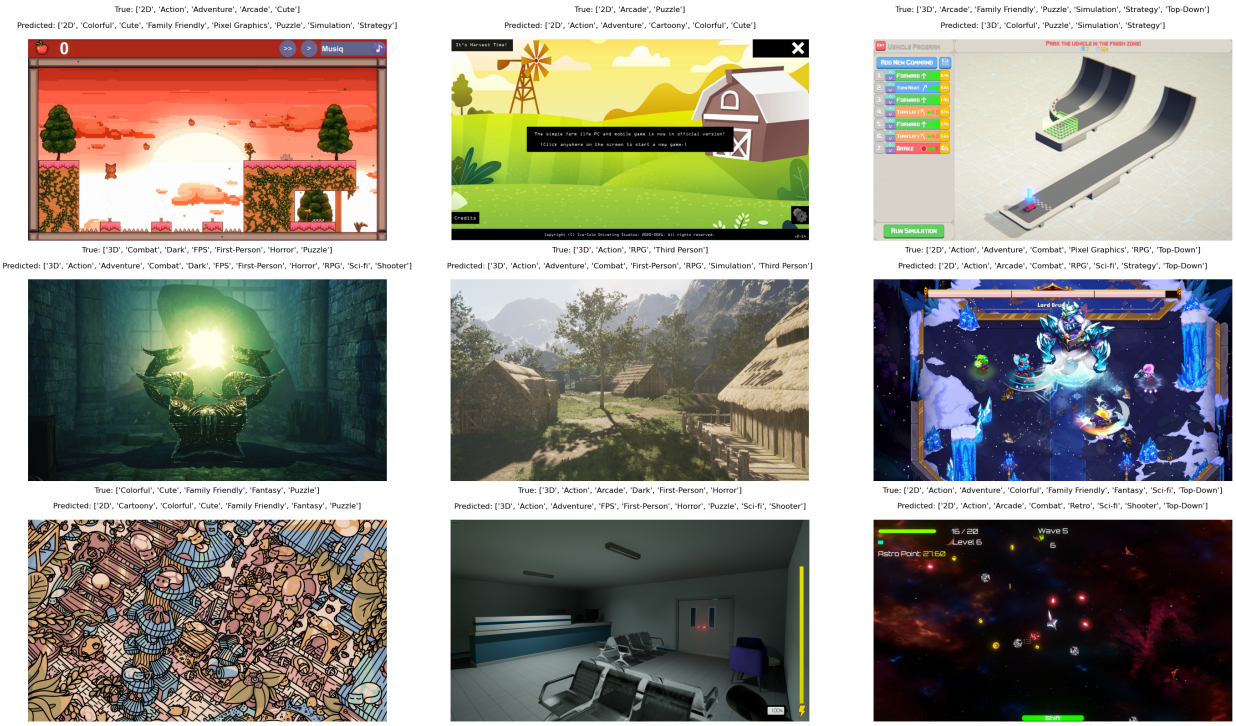
Finally, once the model was chosen, completely optimized and the custom tag thresholds were determined, the model was used to predict the tags from the holdout set. With this set, the model achieved a precision score of 0.46, a recall of 0.59 and an F1 score of 0.51. The results for each tag are summarized in the figure below. Each tag has an individual precision,

recall, F1 score and support value (the proportion of true instances in the dataset, y-axis on the right).



Discussion

Looking at the bar chart above, a couple of trends emerge. Firstly, there appears to be a strong association between how well represented a tag is in the dataset, and its resulting F1 score. This suggests that the model would greatly benefit in accuracy if more instances of these tags were to be present, or if a larger dataset in general would be to used. Secondly, all but one tag have a recall score higher than their respective precision score, suggesting the model struggled slightly with false positives, giving out too many true labels. Overall the accuracy of the model was quite decent, considering the dataset and the classification problem it was trying to solve. Inconsistencies regarding the labelling of the data have already been discussed. To exemplify this, nine screenshots with their corresponding true and predicted labels were randomly selected from the test dataset and are displayed below. We see that the majority of the predicted labels here are actually quite plausible, and may well have been also predicted by a human control. In the top middle and bottom left screenshots, the model predicts the games being tagged as ‘Cartoony’. This is a very fair prediction given the animation style of the game. However, neither of these games are tagged as ‘Cartoony’. This highlights the tag inconsistencies. It should also be noted that another less popular tag ‘Cartoon’ was included in the original list of tags, but was removed on the basis of similarity. These two very similar tags very rarely co-occurred. Similarly, for the bottom right screenshot, the game predicts ‘Arcade’, ‘Combat’, ‘Retro’, ‘Sci-fi’, which are all reasonable predictions given the screenshot, but none of them represent true labels.



There were also many instances where true tags did accurately depict the game, but these tags were simply not represented in that particular screenshot, and so the model would have a very difficult time to predict the tag based on this screenshot alone. One way that would likely greatly improve the reliability and accuracy of model predictions would be to aggregate the predictions of all screenshots from a single videogame, so that one prediction per videogame would be found. This would mean that the model could base its decision off multiple different representations of the game, and would aid with the issue of tags only being represented in one or a few of the screenshots. An even better solution would be to train the model on these aggregations so that it is not penalized for making predictions it should not be expected to make, thus aiding in the final model outcome. However, this would be difficult considering the varying amount of screenshots per game, with a dynamic batch size needed to ensure all screenshots from a given game were contained within the same batch. The output of each screenshot would also need to be changed to the aggregation before backpropagation took place. However, a model trained like this may be able to achieve very high degrees of accuracy, and given a larger dataset, may be able to accurately predict a much wider range of tags.