Deep Learning Assignment 1

Name: Steven Mulkerrins

Student Number: D23127489

Code Link: <https://github.com/smulkerrins/deeplearning_assignment1>

Models Link: <https://github.com/smulkerrins/dlassignment>

Statement: *“I confirm that the document and related code here is all my own work and that I did not engage in unfair practice in any way.”*

# Part 1: Genre Prediction Task

## Dataset

In this part we explore text classification using the “Multi-Lingual Lyrics for Genre Classification” dataset provide on Kaggle (Bejan, 2020).

This dataset contains approximately 290,000 rows, with each row containing data corresponding to an individual ‘song’ or ‘music track’.

The dataset contains multiple features describing each song. For this exercise the relevant features were the Artist, Lyrics and Genre. Other features were ignored for the purposes of this exercise.

## Methodology

### Code

Following some initial experimentation with creating models and training them against the dataset, I determined that individually building, training and evaluating each combination of network architectures and hyperparameters would take more time than was available.

To help with this, I attempted to implement a simplistic ‘grid search’ style framework that would allow me to define a number of model configurations, with each configuration specifying a network architecture and set of hyperparameters. The framework would then automatically build, train and evaluate a model for each configuration specified.

This approach ultimately cost more time than it saved, and in particular made it difficult to make small adjustments to models based on previous outcomes, and I would not recommend it. In hindsight, if this was a viable approach, it likely would already be built into the Tensorflow framework.

### Data Preparation

The dataset is already split into training and testing sub sets. To allow for easy comparison with other experiments on this data I did not perform any further splitting or shuffling of the dataset.

While the dataset was very high quality, there was a small number of rows with nulls or invalid values. These were removed using the Pandas dropna() function.

### Encoding and Padding

The sets of Artists, Lyrics and Genres were all tokenised using the Tensorflow provided Tokenizer function.

I created and fit and individual Tokenizer per feature. For each feature, I used a union of both training and testing data to ensure that the same tokenizer word lists could be used for both training and testing.

For the Genre feature, I applied a filter while tokenizing to filter out special characters. This prevented genres such as R&B or Hip-Hop from being treated as separate tokens. This was not required for the Artist or Lyric features as they are not our target features.

For the Lyrics feature, padding was also applied to regularise the size of the data for embedding.

### Model Compilation and Training

I compiled each model with an initial embedding layer, with the dimensions scaled to match the Lyrics encodings. For models which would also include the Artist feature, I added an additional Embedding layer, and then flattened both.

I then added the required hidden layers for each model according to the network architecture and hyperparameters to be tested.

For output from our model, I added a dense layer with configurable activation function after the hidden layers.

I trained each model using the encoded Lyrics, Genres, and optionally Artist, features over a configurable number of epochs and with a fixed validation split of 0.2.

## Evaluation

For each of our models we will evaluate the training and validation loss and accuracy during their training.

For each model we will then predict against the test image set provided in the CIFAR-100 dataset. Using SciKit Learn’s Metrics library we will generate a classification report and use the Overall Accuracy score to determine the accuracy of our model.

Finally we will render a confusion matrix to visualise the label predictions being made by our model against the true labels.

## Execution

### Model 1: Single LSTM 64 Layer

The first model uses a single hidden LSTM layer of size 64. It uses the adam optimizer, softmax for activation on the output layer and sparse categorical crossentropy as the loss function.

#### Training

|  |  |
| --- | --- |
|  |  |

The training loss decreasing indicates that the model is learning against the training data, however with the validation loss increasing this suggests that the model is not generalising well and is overfitting. This is backed up by the training accuracy increasing with training while the validation accuracy remains almost static.

#### Evaluation

Our model has an overall accuracy of 0.1609. When predicting against 10 classes this suggests that we are doing better a naïve model making random guesses, or a model predicting only a single class, but not by much.

A chart with different colored squares

Description automatically generated with medium confidence

Reviewing our confusion matrix we can see that our model is heavily predicting against the Rock genre, which is also where it’s predictions are most accurate. This suggests the model is learning from an imbalance in the training data.

### Model 2: Single LSTM 128 Layer

The second model also uses a single hidden LSTM layer and increases size to 128. It also uses the adam optimizer, softmax for activation on the output layer and sparse categorical crossentropy as the loss function.

#### Training

|  |  |
| --- | --- |
|  |  |

Again we see divergence of training and validation loss, along with increasing training accuracy and near static validation accuracy, which is very characteristic of overfitting. As we would expect, increasing the hidden layer size did not have any impact on overfitting.

#### Evaluation

Our model has an overall accuracy of 0.1776. This is a small improvement but not by much.

A chart with blue and white squares

Description automatically generated with medium confidence

Our confusion matrix again shows that our model is learning the datasets imbalance toward the Rock genre.

### Model 3: Double LSTM 64 Layer

The third model uses two hidden LSTM layer and of size 64, and also introduces a dropout of rate 0.2. It also uses the adam optimizer, softmax for activation on the output layer and sparse categorical crossentropy as the loss function.

#### Training

|  |  |
| --- | --- |
|  |  |

Our training and validation loss and accuracy are again characteristic of overfitting. Using two hidden layers, even with the introduction of dropout is not preventing overfitting.

#### Evaluation

Our model has an overall accuracy of 0.1791. This improvement is barely measurable.

A chart with blue squares and numbers

Description automatically generated with medium confidence

Our confusion matrix suggests that any improvement in accuracy is only coming from the model increasing it’s number of Rock genre predictions.

### Model 4: Double LSTM 128 Layer

The fourth model also uses two hidden LSTM layer and increases size to 128, and also retains a dropout of rate 0.2. It also uses the adam optimizer, softmax for activation on the output layer and sparse categorical crossentropy as the loss function.

#### Training

|  |  |
| --- | --- |
|  |  |

Our training and validation loss and accuracy are almost identical to the previous model.

#### Evaluation

Our model has an overall accuracy of 0.1725. This is a negligibly small dis-improvement over the previous model.

A chart with blue squares and numbers

Description automatically generated with medium confidence

Our confusion matrix suggests that the dis-improvement is probably due to the model making fewer Rock predictions.

### Model 5: Single LSTM 128 Layer Categorical Cross Entropy

The fifth model uses a single hidden LSTM layer of size 128. It also uses the adam optimizer, softmax for activation on the output layer but uses only categorical crossentropy as the loss function.

I was unable to get this model to train successfully due to a mismatch in the dimensions of data being passed to the output layer which I could not correct in time.

#### Training

N/A

#### Evaluation

N/A

### Model 6: Single LSTM 128 Layer SGD

The fifth model uses a single hidden LSTM layer of size 128. It also uses softmax for activation on the output layer and sparse categorical crossentropy as the loss function, but uses stochastic gradient descent as the optimizer function.

This model provided no appreciable change over earlier models.

### Model 7: Single LSTM 128 Layer RELU

The fifth model uses a single hidden LSTM layer of size 128. It also sparse categorical crossentropy as the loss function, and uses softmax as the optimizer function, but uses relu as the activation function.

This model provided no appreciable change over earlier models.

### Model 8: Triple LSTM 128 Layer

The fourth model also uses three hidden LSTM layer and increases size to 128, and also retains a dropout of rate 0.2. It also uses the adam optimizer, softmax for activation on the output layer and sparse categorical crossentropy as the loss function.

This model provided no appreciable change over earlier models.

### Model 9: Single RNN 64 Layer

The ninth model uses a single hidden SimpleRNN layer of size 64. It uses the adam optimizer, softmax for activation on the output layer and sparse categorical crossentropy as the loss function.

This model estimated that it would require 100 days to train and will be skipped

### Model 10: Single RNN 64 and LSTM Layer

The tenth model uses a single hidden SimpleRNN layer of size 64, with an LSTM layer also of size 64. It uses the adam optimizer, softmax for activation on the output layer and sparse categorical crossentropy as the loss function.

This model also estimated that it would require 100 days to train and will be skipped

### Model 11: Single LSTM 64 Layer 20 Epochs

The eleventh model uses a single LSTM layer of size 64. It uses the adam optimizer, softmax for activation on the output layer and sparse categorical crossentropy as the loss function. This model increases the number of training epochs from 10 to 20.

This model provided no appreciable change over earlier models.

### Model 12: Single LSTM 64 Layer

The twelfth model uses a single LSTM layer of size 64. It uses the adam optimizer, softmax for activation on the output layer and sparse categorical crossentropy as the loss function. This model includes the Artist feature when training and evaluating the model.

This model is again very similar to previous models, but I am including the full details as I was expecting the inclusion of the Artist feature to improve the model somewhat.

#### Training

|  |  |
| --- | --- |
|  |  |

Our training and validation loss and accuracy are almost identical to previous models. Even with the inclusion of the Artist feature, our model is still overfitting.

#### Evaluation

Our model has an overall accuracy of 0.1270. This is a noticeable dis-improvement over previous models.

A chart with blue squares and white text

Description automatically generated

Our confusion matrix does suggest that at least the model has broadened the range of it’s predictions, and inaccuracy is now maybe less about overfitting.

# Part 2: Transfer Learning and Image Processing

## Dataset

In this part we use the CIFAR-100 dataset (Krizhevsky, 2009). The dataset provides a set of 100 classes, containing 600 images each. The dataset is balanced with the same number of training (500) and test (100) images per class. The classes are also grouped into super classes, but we will ignore the super classes for this exercise.

For this exercise, we split the images for use by class into two blocks, referred to hereafter as Block 1 and Block 2. This was done by randomly selecting 50 classes, assigning them to Block 1, and assigning the remaining classes to Block 2.

## Evaluation

For each of our models we will evaluate the training and validation loss and accuracy during their training.

For each model we will then predict against the test image set provided in the CIFAR-100 dataset. Using SciKit Learn’s Metrics library we will generate a classification report and use the Overall Accuracy score to determine the accuracy of our model.

Finally we will render a confusion matrix to visualise the label predictions being made by our model against the true labels.

## Basic Modelling: For Block 1 images

For this task, we trained eight different networks with variations in hyperparameters and network structure against the Block 1 images, and evaluated each to identify the best approach

### Model 1: Basic model

Following some initial experimentation, we settled on a basic model consisting of three convolutional layers for feature extraction. This provides a good balance between allowing for more complex feature detection in image classification, without adding unnecessary complexity given the relatively small and simple nature of the CIFAR-100 images.

We included five dense hidden layers for classification, starting with ReLU activation as a good default for image classification problems (Goh, 2024).

We then add a single dense output layer using softmax activation, as it is very efficient at classification for large numbers of classes (Zharmagambetov, 2024).

#### Training

|  |  |
| --- | --- |
| A graph of training and validation loss  Description automatically generated | A graph of a line graph  Description automatically generated |

The increase of both training and validation loss, with the decrease of training and validation accuracy suggests either an incorrect learning rate, loss function, or optimiser. Give the instability of the increases and decreases it may also suggest an ‘exploding gradient’ problem.

#### Evaluation

This model scored an overall accuracy of 0.0226. Given we are predicting against 50 classes, this is roughly equivalent to a naïve model making random guesses, or guessing a single class each time.

A screen shot of a graph

Description automatically generated

Reviewing our confusion matrix, we can in fact see that the model is mostly predicting a single class, with a small number of predictions against a second class.

### Model 2: Data Augmentation

For the next model, we attempted to include Data augmentation, using the Tensorflow ImageDataGenerator function to generate additional images for training. All other hyperparameters and network structure were kept the same.

#### Training

|  |  |
| --- | --- |
|  |  |

The training and validation loss was negligible in early epochs, spiking hugely in the final two epochs, suggesting some issue with how the data was batched. This is likely caused by the data augmentation used. Training and validation accuracy remained low, collapsing at the end of training, again likely to do with a batching issue caused by the data augmentation.

#### Evaluation

This model scored an overall accuracy of 0.0200. This even more closely matches a naïve model making single class predictions.

A graph with different labels

Description automatically generated with medium confidence

This is born out in our confusion matrix with the model even more heavily favouring a single class with only a small number of predictions against a second class.

### Model 3: Increased training epochs

For the next model, removed the Data augmentation, and experimented with increasing the number of training epochs. All other hyperparameters and network structure were kept the same.

#### Training

|  |  |
| --- | --- |
|  |  |

The training and validation loss and accuracy were both unstable, increasing and decreasing, pointing back to the issues identified with the original model.

#### Evaluation

This model scored an overall accuracy of 0.0350. This suggests a very small improvement over a naïve model.

A graph with blue and white text

Description automatically generated with medium confidence

We see this in our confusion matrix with the model still heavily favouring a single class but with slightly more predictions against multiple other classes.

### Model 4: ReLU activation vs eLU activation

For this model we changed the hidden dense layers to use eLU activation rather than ReLU activation. All other hyperparameters and network structure were kept the same.

#### Training

|  |  |
| --- | --- |
|  |  |

The training and validation loss now have a noticeable trend downwards across epochs, suggesting that the model is learning more across each epoch, until a very minor increase at the last epoch suggesting we are potentially getting close to overfitting. Training and validation accuracy also both increase across each epoch, but with training accuracy continuing to increase and validation accuracy slightly decreasing on the final epoch this validates that our model is in fact learning, but coming close to overfitting.

#### Evaluation

This model scored an overall accuracy of 0.4372. A huge improvement over a naïve model.

A graph with blue squares

Description automatically generated

Our confusion matrix backs this up showing a clear diagonal line indicating a large number of successful predicted labels against true labels.

### Model 5: Leaky ReLU

For this model we changed the hidden dense layers to use Leaky ReLU activation. All other hyperparameters and network structure were kept the same.

#### Training

|  |  |  |
| --- | --- | --- |
|  |  |  |

The training and validation loss maintain the noticeable trend downwards across epochs, again suggesting that the model is learning more across each epoch. Training and validation accuracy also continue to both increase across each epoch. Indications of possible overfitting are similar to the eLU model.

#### Evaluation

This model scored an overall accuracy of 0.4420. A huge improvement over a naïve model, and slight improvement over the eLU model.

A graph with blue squares

Description automatically generated with medium confidence

Our confusion matrix is again similar to the eLU model with maybe a slight improvement noticeable.

### Model 6: Logistic Activation

For this model we attempted to implement logistic activation in the final layer through the Sigmoid function. Unfortunately we were not able to construct a working model due to mismatches in the dimensions of outputs being passed from the hidden dense layers. We were not able to resolve this mismatch in the time available.

#### Training

N/A

#### Evaluation

N/A

### Model 7: Skip Connections

For this model we implemented skip connections. Skip connections are a powerful tool for improving image classification networks, allowing gradients to flow more easily through the network (O'Neill, 2021).

#### Training

|  |  |  |
| --- | --- | --- |
|  |  |  |

The training loss peaked early during training but then remained quite constant. Validation loss remaining quite constant across training. Training accuracy trended upwards across training, however validation accuracy was unstable suggesting little or no real learning was happening.

#### Evaluation

This model scored an overall accuracy of 0.0902. While this is a clear improvement over a naïve model, it is substantially behind both the Leaky ReLU and eLU models.

A graph with blue lines and white text

Description automatically generated with medium confidence

Our confusion matrix also suggests that the model may be performing better for some classes than others, though it lacks the clear diagonal which would suggest it is learning across all classes.

### Model 8: Increased Epochs & Early Stopping

For this model we re-implemented our current best performing model (Model 5: Leaky ReLU), increased the number of training epochs allowed, but also added early stopping to allow the model to stop training when it is no longer learning and to protect from overfitting.

#### Training

|  |  |  |
| --- | --- | --- |
|  |  |  |

As with our previous Leaky ReLU model, we see loss trending downwards and accuracy trending upwards as hoped. However this time the model can continue training until there is no progress made and early stopping kicks in.

#### Evaluation

This model scored an overall accuracy of 0.4150. This is a good score and again similar to both the Leaky ReLU and eLU models.

A graph with blue squares

Description automatically generated

Our confusion matrix again has tell-tale diagonal line previously seen in both the Leaky ReLU and eLU models.

## Autoencoder Modelling: For Block 1 Images

For this exercise, I selected Model 8 from the previous classification exercise to act as the base for an autoencoder model.

To create the encoder I took the first five layers, the convolutional and flattening layers, from this model. This transfer learning will allow our autoencoder to recognise features which have been learned by our classification model.

To construct the decoder I added four Conv2DTranspose layers to learn how to mirror and invert the encoder, with each layer learning how to upsample and reconstruct features. In between these layers I included UpSampling2D layers to upsample the feature maps for the next layer.

I then combined both the encoder and decoder in a sequential model to complete the autoencoder.

#### Training

|  |  |  |
| --- | --- | --- |
|  |  |  |

The training and validation loss steadily decreased in a nice curve across all 10 training epochs.

#### Evaluation

To evaluate the autoencoder, we applied the test set of images from the CIFAR-100 dataset, and then calculated the Mean Squared Error (MSE) between the generated images and the real test images. An MSE of 0 would indicated that the generated images exactly matched the real images. The higher the MSE value, the more our generated images would diverge from the real images

The MSE of our autoencoder was calculated at 0.008662259206175804, this is very close to the ideal score of 0, and indicates that our autoencoder is generating images close to, but not exactly matching, the real test images.

A visual comparison of a sample of original images from our test set against generated images shows a very close match. While the generate images are clearly not identical, they strongly resemble the originating image. Features from the originating image are clearly recognisable, and reconstructed images can be easily matched with their originating images and would not be confused for other originating images. In fact the generated images look very much like a down-sampled copy of the originating images. A future experiment to increase the amount of up-sampling in the decoder may give even better reproductions

A blurry image of a dog

Description automatically generated

A close up of a person's face

Description automatically generated

A comparison of a blurry image

Description automatically generated

A collage of images of apples

Description automatically generated

A comparison of a dog's face

Description automatically generated

## Transfer Learning for Block 2 Images

### Initial Transfer Model

A brand new model was trained from scratch on the Block 2 images to act as a control. This model was trained using the same network architecture and hyperparameters as the previous best model from the classification exercise. This model will be referred to as the ‘from-scratch model’.

To demonstrate transfer learning, I took the first five convolutional and flattening layers from the previous best model from the classification exercise. These layers were then frozen, setting their trainable property to false. I then added this model to a new transfer model. On the transfer model, I added three new dense hidden layers, with Leaky ReLU activation as before. The model was then trained on the Block 2 set of images from the CIFAR-100 dataset. This model will now be referred to as the ‘transfer model’.

Finally, additional fine tuning was applied to the transfer model, unfreezing three of the initial convolutional layers, and retraining on the Block 2 set of images.

### Model 1: From-Scratch Model

For this model we re-implemented our current best performing model (Model 8: Leaky ReLU) from image classification. This time the model was trained on the Block 2 set of images.

#### Training

|  |  |  |
| --- | --- | --- |
|  |  |  |

As with the Block 1 set of images, we see loss trending downwards and accuracy trending upwards as hoped, demonstrating good learning for our model across each epoch.

#### Evaluation

This model scored an overall accuracy of 0.4150. Our from scratch model is performing as expected and very similarly to the Block 1 set of images.

A screen shot of a graph

Description automatically generated

Our confusion matrix again has tell-tale diagonal line previously seen in for the Block 1 images.

### Model 2: Transfer Learning Model

For this model we used the convolutional layers previously trained on Block 1, adding new hidden layers to be trained on the Block 2 images.

#### Training

|  |  |  |
| --- | --- | --- |
|  |  |  |

Both training and validation loss increase rapidly. This indicates significant overfitting. This is likely to be a direct result of freezing the convolution layers trained to identify features in Block 1 images which are not present in the Block 2 images

#### Evaluation

This model scored an overall accuracy of 0.0200. This is exactly what we would expect from random chance or from the model only predicting a single class.

A graph with a blue line

Description automatically generated with medium confidence

Our confusion matrix confirms that the model is only predicting against a single class.

### Model 3: Fine Tuning the Transfer Learning Model

For this model we unfreeze the first 3 convolutional layers previously trained on Block 1, allowing them be trained on the Block 2 images to hopefully learn any Block 2 specific features.

#### Training

|  |  |  |
| --- | --- | --- |
|  |  |  |

Both training loss decreases rapidly, hopefully indicating that we are now learning Block 2 specific features. However training accuracy is unstable, while validation accuracy remains locked at 0.02 indicating that no useful learning is happening

#### Evaluation

This model scored an overall accuracy of 0.0200. Again this is exactly what we would expect from random chance or from the model only predicting a single class.

A graph with text on it

Description automatically generated with medium confidence

Our confusion matrix again confirms that the model is only predicting against a single class.

### Conclusion

Both the transfer model and fine tuned model failed to demonstrate any useful learning or predictive ability. This may be because the source model used for transfer learning has been overfitted to the Block 1 set of images.

It is also possible that accurate image classification requires learning a distinct, non-transferrable set of image features related to a given label and that image classification is not a good candidate for transfer learning. A set of image features used to identify a ‘Fish’, may not be relevant to the features needed to identify a ‘Porcupine’.

A brief review of existing related literature (Plested, 2022), suggests that training against fully labelled datasets should always be the preference, but that transfer learning is helpful when fully labelled data is not available. This suggests that our transfer and fine tuned models should have performed better, though not as well as the from-scratch model.

# Bibliography

Bejan, M. (2020). *Multi-Lingual Lyrics for Genre Classification* . Retrieved from Kaggle: https://www.kaggle.com/datasets/mateibejan/multilingual-lyrics-for-genre-classification

Goh, K. S. (2024). *Emerging Science Journal*. Retrieved from Comparison of Activation Functions in Convolutional Neural Network for Poisson Noisy Image Classification: https://doi.org/10.28991/esj-2024-08-02-014

Krizhevsky, A. (2009). *University of Toronto*. Retrieved from Learning Multiple Layers of Features from Tiny Images: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

O'Neill, D. X. (2021). *Evolutionary Neural Architecture Search for High-Dimensional Skip-Connection Structures on DenseNet Style Networks*. Retrieved from IEEE Transactions on Evolutionary Computation: https://doi.org/10.1109/TEVC.2021.3083315

Plested, J. &. (2022). *Deep transfer learning for image classification: a survey.* Retrieved from https://arxiv.org/abs/2205.09904

Zharmagambetov, A. G.-P. (2024). *Softmax Tree: An Accurate, Fast Classifier When the Number of Classes Is Large*. Retrieved from ACL Anthology: https://doi.org/10.18653/v1/2021.emnlp-main.838