Generating Lyrics with Machine Learning

Dan Waters - CSCE 5218 - Spring 2021

Note: The original proposal described a different topic (GAN-based image generation).

Abstract

What happens when you start having fun with Machine Learning? This project happens. I was initially planning to generate "custom dogs," but decided instead to go with an NLP problem.

There are multiple ways to generate strings of text: transformers, autoencoders, VAEs and so on, but to understand the basics, let's explore recurrent neural networks and LSTM architectures.

Given a large amount of text representing a large portion of an artist's lyrics, can we train a model to predict the next word or character given some input string? Will the text look like English? Will it be grammatically correct? These are all questions to answer with this project.

Data set

The data comes from a Kaggle dataset called Song Lyrics[1]. I experimented with multiple different artists, mostly with hilarious results in the beginning. The Eminem song my model generated will not be shared in this academic work.

I augmented the dataset with some other secret artists which I may incorporate later.

All of the code and training data for this project is available on GitHub[2].

Approach

I investigated three different, but related, methods for text generation. From the available dataset, I used Michael Jackson for most of my experiments due to the large vocabulary and relative appropriateness for publication (as opposed to Eminem or Kanye West). The source file has about 11,000 lines of his lyrics.

Method A: RNN with Character Embeddings

The first method I investigated was a recurrent neural network with state information passed into each prediction, based on a tutorial from the official TensorFlow documentation for RNNs[3]. The first iteration of this model has the following architecture (**Fig. 1**):

---

[1] "Song Lyrics | Kaggle." *Kaggle, 2018, https://www.kaggle.com/paultimothymooney/poetry*

[2] "danwaters/lfi: lyrics from image." *GitHub, 25 April 2021, https://github.com/danwaters/lfi*

[3] "Text generation with an RNN | TensorFlow Core." *TensorFlow*, Google LLC, 02 April 2021, https://www.tensorflow.org/tutorials/text/text_generation

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        multiple                  23808
_________________________________________________________________
gru (GRU)                    multiple                  1182720
_________________________________________________________________
dense (Dense)                multiple                  47709
=================================================================
Total params: 1,254,237
Trainable params: 1,254,237
Non-trainable params: 0
_________________________________________________________________
```

Figure 1. RNN with Character Embeddings Architecture.

The embedding layer is the input layer, and in this example, maps character IDs to 256-dimensional vectors. TensorFlow's `preprocessing.StringLookup` is used to navigate and maintain the lookups.
As this is not a classification task, the only available metric is loss on the training data, and we also do not hold out any validation or test sets for this task.

Prediction is run in a loop, calling the model with updated internal state every time. Because we are working at the character level, the model often generates totally nonsensical (and hilarious) words. This approach is simple and entertaining, but it is not good enough to maintain lyrics. It might be funny in a meme context, but it isn't very convincing. (**Fig. 2**)

```
Ach you leff on wrifpln't me hay
Hever a chat of feel be? I nee you ge)
You to be reel, dan a willo you know shat
tries on My Be
Ooh Qo Vo, You
Taind Orres
To Mate You can't But Of, Ux5)
I Ta seee to stant Up any
Romey horge and frazt my how you reem
Your loofin' in!
```

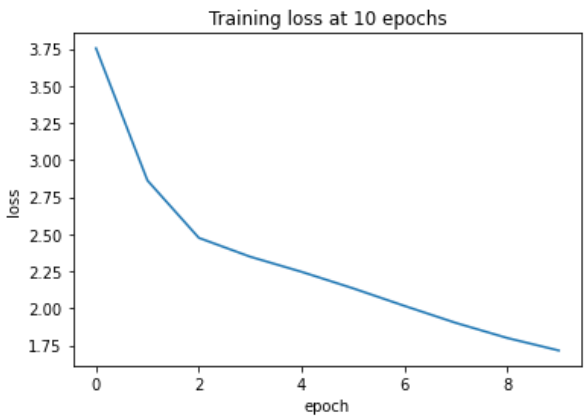


Figure 2: RNN generating Michael Jackson lyrics after 10 epochs of training.

At ten epochs, there are many words and punctuations that don't make sense at all. It's also obvious the network has not learned the concept of open and closed quotes and parentheses. Let's keep going. Thankfully, this model trains very quickly with a GPU enabled.

At 25 epochs, the model is considerably better. My favorite lyrics from this song are "When't me spece" and "Monsevery heaven." (**Fig. 3**)

```
I jadala, papers
Life it, feeling like a back in a mminbul
trusble
Tell me what you have me?

I'll conightring ip you love
I don’t know, know in her man ie and love
and shout
Move that they really hoou
Brothine
When't me spece
I'm with the force don't stop
Don't stop, I'll beau
With you once again
Monsevery heaven
```

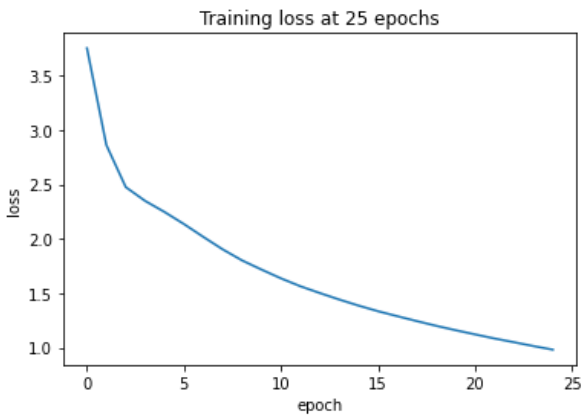


Figure 3: RNN-generated Michael Jackson lyrics after 25 epochs of training.

At 100 epochs, it is looking better in some ways and worse than others. The loss improvements are also tapering off. (**Fig. 4**)

```
I Got To Know Why
There's A Love For Me
There Are People Dying
If You Care Enough
For The Living
Make A Better Place
For You And For Me

Heal The World
Make It A Better Place
For You And For Me

[Michael]
Hoo!
(Who Is It?)
I Believe In Me
So You Feel Slack Time
If You Can't Feed Your Baby, Ye
```

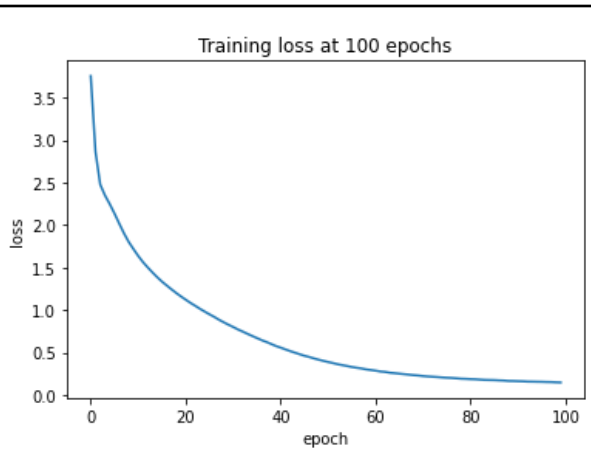


Figure 4: RNN-generated Michael Jackson lyrics after 100 epochs of training.

There is no notable change between the 150 epoch lyrics and the 100 epoch lyrics. The actual loss difference here was miniscule; diminishing returns really kicked in around the 80th epoch. Still, the lyrics are funny. (**Fig. 5**)

```
I'm in love
(love)
I'm in love with a beautiful girl
I'm in love with a beautiful girl
I'm in love with a boom
Who is the teres to me
But she love him some

Now must there a read of be
I'm a dogning in the world down
They just couldn't take it
They every step of the way yeah yeah
An is bad, I'm bad - I'm bad baby
You gotta touch the one you love
If you want the one you love to touch you
You're my sidgen pinder
```

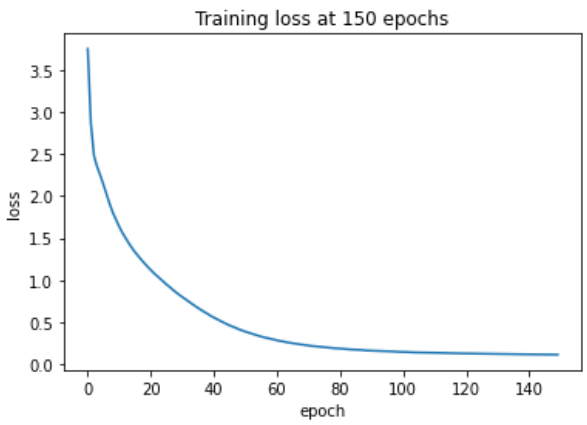


Figure 5: *RNN-generated Michael Jackson lyrics after 150 epochs of training.*

The documentation says we can also use LSTM instead of GRU layers for this model. Let's try that and see if there is a notable difference.

Method B: LSTM with Character Embeddings

Keeping everything else the same, I wanted to see how an LSTM layer performed instead of the GRU. Training definitely took longer - the LSTM clocked in at about 4 seconds per epoch, compared to around 1 second for the GRU layer.

Like the model described in Method A, this model has 1.2 million parameters: (**Fig. 6**)

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      multiple                  23808
_________________________________________________________________
gru_1 (GRU)                  multiple                  1182720
_________________________________________________________________
dense_1 (Dense)              multiple                  47709
=================================================================
Total params: 1,254,237
Trainable params: 1,254,237
Non-trainable params: 0
_________________________________________________________________
```

Figure 6: LSTM model architecture with character level embeddings.

Immediately, I noticed the Epoch 10 results were much different than the Epoch 10 results using the GRU. Key observations about this sample include: apparent awareness of opening and closing parenthesis, far less non-words, and it seems... oddly poetic. (**Fig. 7**)

```
I keep kidding around
Like a child, but pardon me...

People say I'm not okay
'Cause I love such every day
But whisper the price of fame, you'll see
Pash doh making for me

[Chorus]

All I know it

That about flower (I want to be free)
To fly away just like a sparrow
(I wanna be free) (Ah, ah, uh, oh baby)
To take my time wherever I go
(Wherever I go)
```

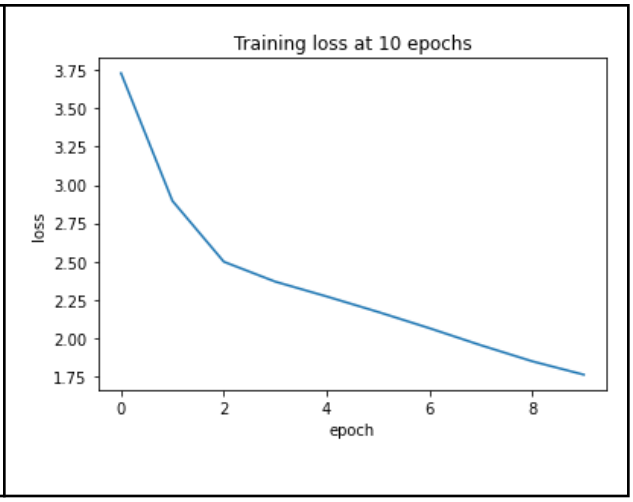


Figure 7: LSTM-generated Michael Jackson lyrics after only 10 epochs of training.

Epoch 25 makes me think we just got lucky on Epoch 10. Loss is still steadily decreasing. (**Fig. 8**)

```
Hurry, hurry mister postman, take my
letter tell her i came he conlr stop
cried
Chearen get me
How we're almost there
Keep on reachin'
For lond stoness
That you give me strong hot fever
My Brother Love, Let Yo Got To Ma
You Can Keep The Baby
That Are We Doin'

To Try To Start My Baby Cryin'

[Bridge]
No Matter How It Starts It Ends The Same
```

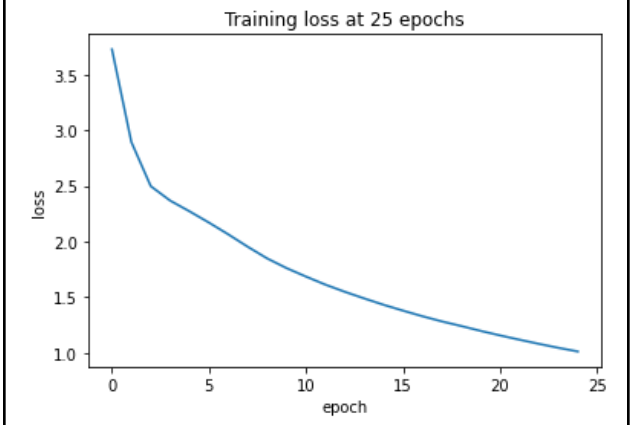


Figure 8: LSTM-generated Michael Jackson lyrics after 25 epochs of training.

Epoch 100 features some oddly suggestive lyrics, and like the other models, seems to be leveling off on the loss reduction at this point. (**Fig. 9**)

```
I'll be showing you what other men are
supposed to do for you my baby

[RAP]

[Chorus x3]

Hold my hand, feel the sweat, yes you do
Right to keep you warm when the night
winds blow
Uu love is maming when I felt down to my
body
there's no douch so ex be fore
Where do you e know
('Cause we're almost there)
Can't you see that you'll be free
To tunth that wonder in my gon'
To find the truth
```

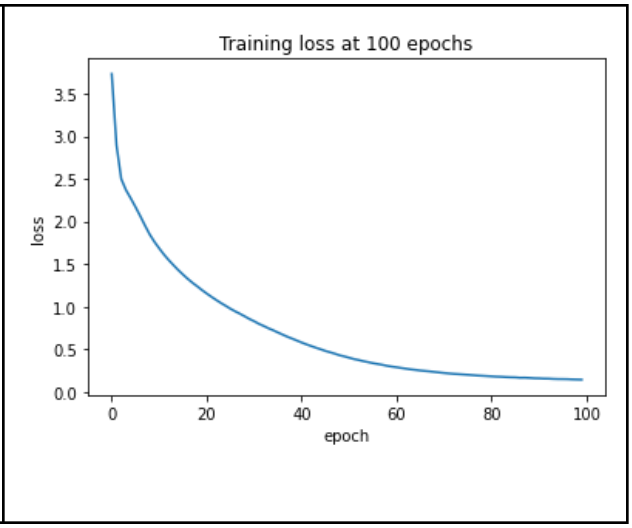


Figure 9: LSTM-generated Michael Jackson lyrics after 100 epochs of training.

The final results at Epoch 150 have not improved much, neither by intelligibility or loss minimization. (**Fig. 10**)

```
I keep kidding around
Like a child, baby
Heaven get readous we sait faysants
attram
Snam heartbreak out off to call
Take me happy and lost my world,
We've got hor fure
Don't understand anyone having the blues
Euphoria never kilding me now
I don't wanna see you in his aron

Come on back to me Maria
Oh come on back to me girl
Hey hey Maria
Maria it's been loving you
(I love ya)
Your love can happened to strow
Uuuuuuh child should have some
othorshound
```

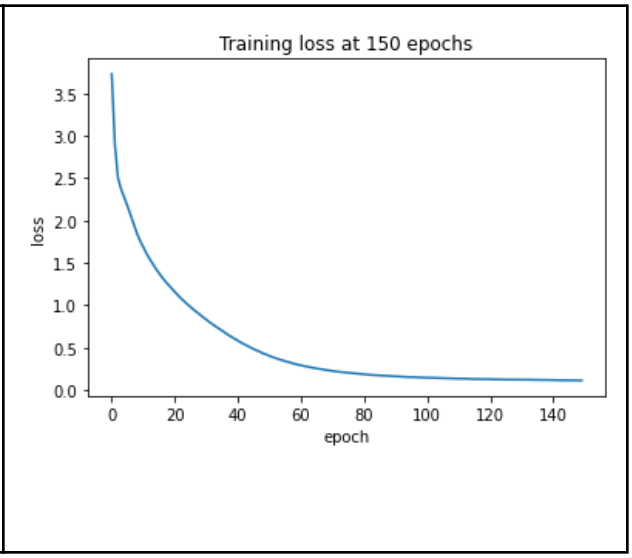


Figure 10: LSTM-generated Michael Jackson lyrics after 150 epochs of training.

Observations

Both the GRU-based and LSTM-based techniques suffer from similar problems in their shared approach:

- The text is predicted at the character level, based on what the model thinks should come next. The "vocabulary" is the alphabet and symbols, rather than words, so nonsensical sequences are generated.
- Some lyrics in the training data have the first letter of every word capitalized, and the model begins overfitting to those cases. Lowercasing the training data should fix that.
- While both models learn from previous information, it has no idea of the concepts in the lyrics. It is just randomly generating text.

In this case, achieving a perfect reconstruction accuracy is actually not desired. We want the model to make some decisions that are amusing and creative. The looping mechanism is wrapped in a class called `OneStepModel`, which includes the argument `temperature` in its constructor. Setting the temperature to 1.0 causes the model to make bolder choices, whereas setting it to 0.1 will cause it to repeat a sequence from that exact song almost perfectly and get stuck there.

Comparing these two methods now is useful, because the third approach is completely different. Training time and minimum loss are quantitative metrics; the rest are qualitative based on the generated results as interpreted by the author. Overall, the LSTM wins this round (unless training time is a consideration). (**Fig. 11**)

	GRU	LSTM
Training time	1 sec / epoch	4 sec / epoch
Minimum loss	0.123 (epoch 150)	0.111 (epoch 150)
Punctuation accuracy	Low	High
Symbol matching memory: (), [], etc.	Low	Good
Nonsense factor	High	Medium

Figure 11. Table containing some legitimate and some rather subjective comparison criteria.

Method C: Bi-Directional LSTM with Word Embeddings

This model is based on [this article from Towards Data Science](https://towardsdatascience.com/nlp-text-generation-through-bidirectional-lstm-model-9af29da4e520)[4], with a couple of important changes. The final model architecture is defined as: (**Fig. 12**)

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_6 (Embedding)      (None, 19, 100)           327400
_________________________________________________________________
bidirectional_1 (Bidirection (None, 19, 300)           301200
_________________________________________________________________
dropout_1 (Dropout)          (None, 19, 300)           0
_________________________________________________________________
lstm_5 (LSTM)                (None, 100)               160400
_________________________________________________________________
dense_4 (Dense)              (None, 1637)              165337
_________________________________________________________________
```

[4] "NLP: Text Generation through Bidirectional LSTM model." *Towards Data Science, 30 Jan 2021, https://towardsdatascience.com/nlp-text-generation-through-bidirectional-lstm-model-9af29da4e520*

```
dense_5 (Dense)              (None, 3274)              5362812
=================================================================
Total params: 6,317,149
Trainable params: 6,317,149
Non-trainable params: 0
_________________________________________________________________
```

Figure 12. Bi-directional LSTM model architecture.

For this model, I did capture accuracy as well as loss. After 100 epochs I was only able to achieve 66% accuracy using this model. (**Fig. 13**)

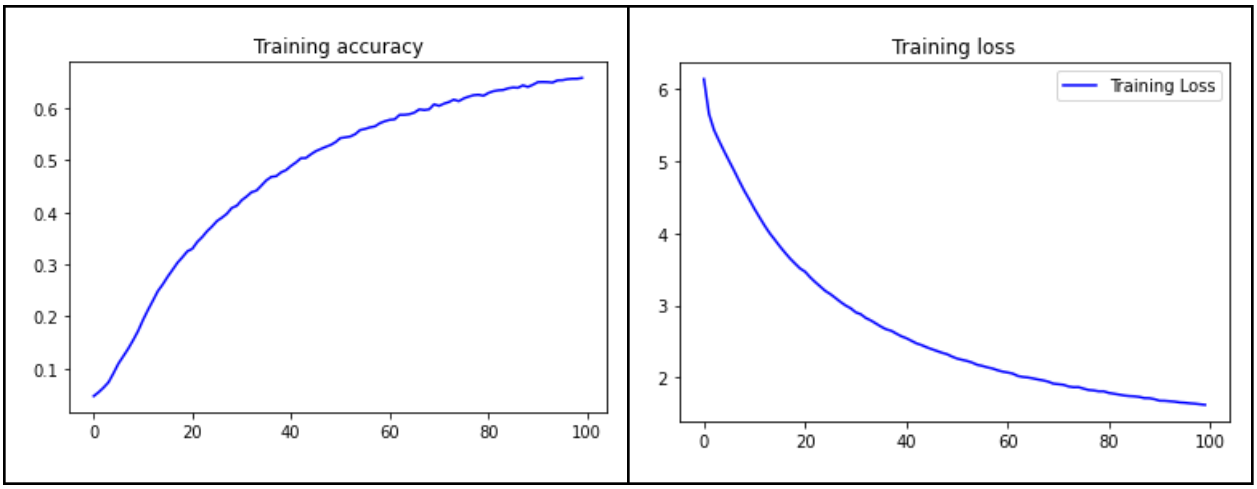


Figure 13: Training accuracy & loss for the bidirectional LSTM model trained on Jackson lyrics over 100 epochs.

I made a couple of changes to this model. First, I updated `predict_classes` to use `np.argmax` on the text matrix instead. This is not a functional change, just an update. The second change is in the sequence that is fed to the network on every inference.

As the source article mentions, the prediction is based upon a growing input sequence that has the predicted word appended to it for every iteration. This results in a long string of text which becomes completely useless after ten words or so; notice how the coherence drops and the prediction destabilizes: (**Fig. 14**)

```
she told me her name was
 billie jean as you're a
 child for such an hour
 is still i change life
 life baby change life life
 and me sweet mouth desire
 life change the soul just
 to escape it gets blown
 da da dum mumble hold
```

```
my hand hold my hand
yeah hold my hand hold
my start change love you
sweet soul life look start
change world my soul desire
world life love letters love
hold your soul world no
devil hold my soul desire
world life life cry sweet
start smile start cry sweet
cry my soul desire love
life baby what life love
```

Figure 14: Bidirectional LSTM predictions, feeding the full seed text to the model to generate words.

This did not meet my expectations; my original hypothesis was that a properly tokenized word embedding would immediately perform better than the by-character recurrent neural network architecture.

My second change was to batch predict sentences using a sliding input rather than the full input string. Once the string is n words in length, the seed text will remain n words long, but dropping the earliest word in the sequence. This stabilized the output, but still resulted in some strangeness. This feels similar to the game where you only hit the middle option on your iPhone predictive text suggestions: (**Fig. 15**)

```
she wants to give it to me
 let me be your someone she
 says i am the one who
 came when you taught you anything
 with love for all the force
 of pain seem cold outside and
 make a vow to take me
 back please write me this way
 you're lovin me' ain't about it
 girl you gotta leave that nine
 to five upon the shelf i
 can't give sweet love to you
 but melt you again i know
 that i would be my father
 never lies price of fame it
 again i know that i would
 be my father never
```

Figure 15: Bidirectional LSTM predicting text with a sliding input of the six most recent words and a seed of the word "she."

One downside of this method is the presentation; it is difficult to know when to insert a new line, so I just chose to insert one every *n* words (where *n* is also the sequence length fed back to the model with each prediction). Surely this task could be completed within this model, but I chose to classify that as "out of scope" for this project.

Conclusions (Natural Language generation)

Based on a purely subjective analysis by this paper's author (i.e. human acceptability of the output), the bidirectional model with word embeddings seemed to generate the most believable lyrics, while the LSTM trained with character embeddings was mostly just hilarious. The bidirectional LSTM did have a tendency to get caught in a bunch of "yeahs." Perhaps an attention mechanism could be used to reduce the priority of that word in a future enhancement.

Bonus: Image Classification

Thanks to the gift of extra time, I was able to explore image classification as well. See the Google Colab notebook for this module.

This component is used as the input to the bidirectional LSTM selected in the earlier phase of the project. Supplying this model with an image will classify the artist, and then generate fake lyrics based on that artist's work by pointing it to an instance of the model trained on that artist.

I built a fairly basic CNN to recognize the four different artists supported by my lyric generation models (The Beatles, Bob Dylan, Michael Jackson, and Frank Zappa). I collected a small dataset of about 50 images per artist, 100 for Frank Zappa. The model architecture is identical to the basic TensorFlow example[5]: (**Fig. 16**)

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling (Rescaling)        (None, 180, 180, 3)       0
_________________________________________________________________
conv2d (Conv2D)              (None, 180, 180, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 90, 90, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 90, 90, 32)        4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 45, 45, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 45, 45, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 22, 22, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 30976)             0
_________________________________________________________________
dense (Dense)                (None, 128)               3965056
```

[5] "Basic classification: Classify images of clothing." *TensorFlow, 19 March 2021, https://www.tensorflow.org/tutorials/keras/classification*

```
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 516
=================================================================
Total params: 3,989,156
Trainable params: 3,989,156
Non-trainable params: 0
_________________________________________________________________
```

Figure 16: Vanilla CNN architecture from the TensorFlow tutorials.

However, training this network for 100 epochs resulted in a validation accuracy of only 56% and a very strange confusion matrix. The model is clearly overfitting to the label Frank Zappa. (**Fig. 17**)

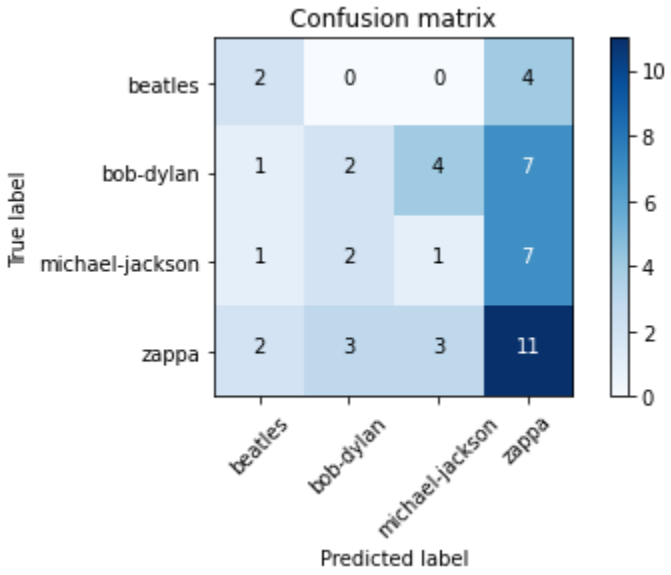


Figure 17. Overfitting to one class.

After training for an additional 100 epochs, a validation accuracy of 0.54 was reached and validation accuracy never really went down, so I decided to add one more Conv2D layer with 128 filters and one more max pooling layer.

The confusion matrix improved a small amount with 100 epochs, but the validation loss stayed static at 0.56: (**Fig. 18**)

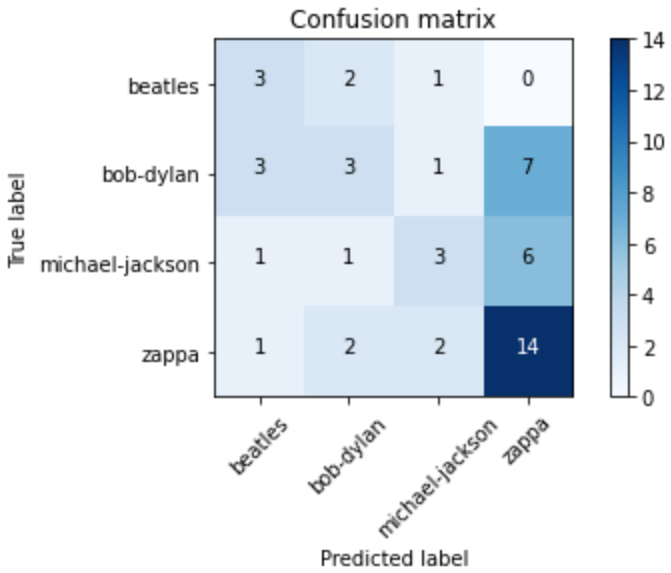
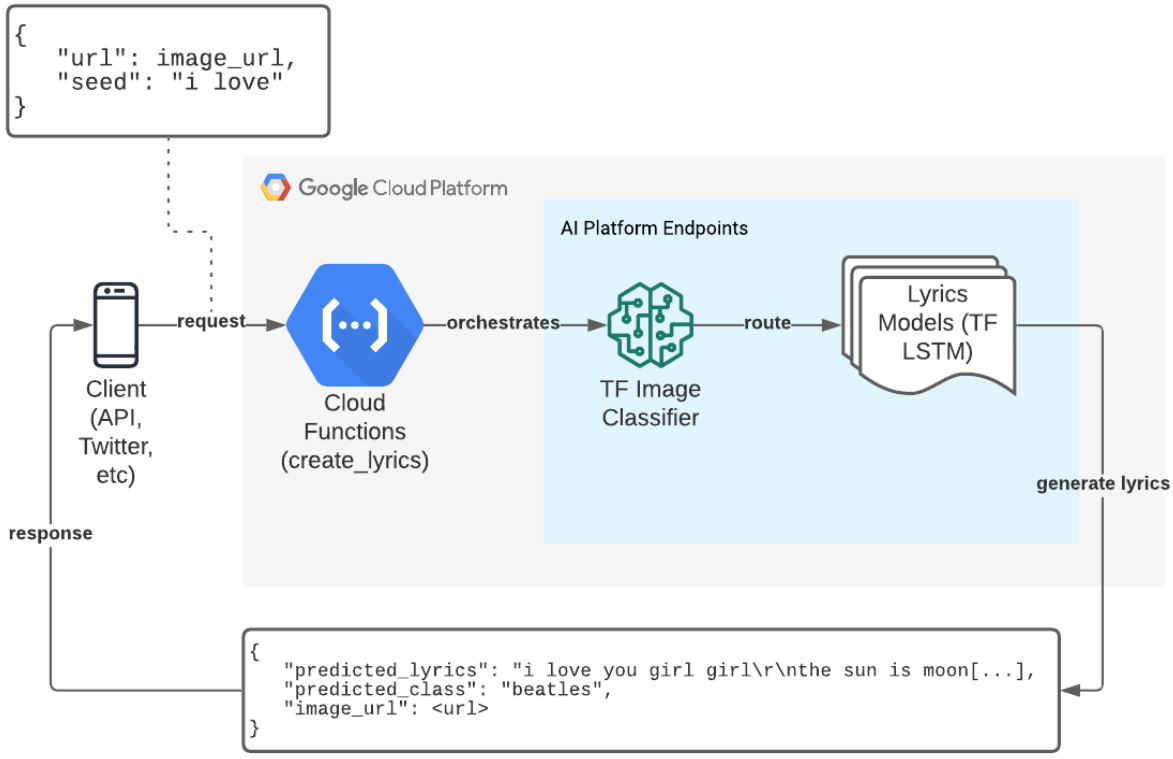


Figure 18: Some improvements in the confusion matrix from an additional filter layer.

My conclusion is that the overfitting condition is a training data problem. There are only a handful of validation images, and the loss is very bouncy. With more data and continued hyperparameter tuning, there is a lot of improvement to expect here.

Bonus 2: It Works

I also decided to see if I could get this to work end-to-end. I pre-trained all of my models in Google Colab, saved the model and weights in SavedModel format, placed those on Google Cloud Storage, exposed them as endpoints through Google's AI platform, and then wrote a Cloud Function (GitHub) to handle the orchestration.



To see this in action, you can use Postman or curl.

How to Test

URL: https://us-central1-dogbot-298321.cloudfunctions.net/create_lyrics

Method: POST

Headers: Content-Type: application/json

Body: (application/json)

{"url": "https://<your_image_url>", "seed": "<text to start the song with>"}

Example response:


```
{
    "image_url":
"https://www.rollingstone.com/wp-content/uploads/2018/06/rs-7349-20121003-beatles-1962-624x420-1349291947.jpg?resize=1800,1200&w=1800",
    "predicted_class": "zappa",
    "predicted_lyrics": " I his gonna along around all me\r\n twirly see up   can\r\n doggie speak tu going long like\r\n in a   might speak\r\n his gonna crop get now i\r\n his gonna along around all me\r\n twirly see up can\r\n doggie speak tu going long like\r\n in a   might speak\r\n his gonna crop get now i\r\n his gonna along around all me\r\n twirly see up   can\r\n doggie speak tu going long like\r\n in a   might speak\r\n his gonna crop get now i\r\n his gonna along around all me\r\n twirly see up "
}
```

Conclusions and Observations

Natural Language

- The Bi-directional LSTM with a sequence count of six had the best performance in the Colab notebook. When deployed, I have questions about whether or not I deployed the model correctly.
- Because AI Platform (unified) does not yet support custom inference code, I had to call the model over REST multiple times to generate each word. This is not just non-performant, it's also very expensive. I would like to explore if there is a way to have the model feed itself for some fixed period and then generate the output. This was one area where I struggled a lot.
- I would also like to see if I could get one model to generate lyrics instead of creating multiple models trained on each dataset. The latter option would not scale with a large number of artists.

Image Classification

- As a secondary objective, I didn't spend as much time here as I did on the RNN and LSTM, but I did learn that a tiny dataset will result in lots of confusion and overfitting.
- I would like to add more labels and more supplementary training data, perhaps using the augmentation techniques we learned in class.

Deployment

- Consolidating prediction routines as much as possible will improve performance and cost.
- Having a local test environment like a functions emulator would drastically speed up development time (iterations, waiting for deployments, testing with the logs). I should have configured that earlier in the project.

Improvements and Enhancements

- Whether the cloud function is working the same way as the NLP Colab notebook is totally debatable. I will investigate it when I have time.
- Image classification accuracy and training dataset size, improve confusion and reduce overfitting.
- Expand image classification accuracy to include all the supported NLP labels.
- Classify the sentiment of the input image and generate lyrics according to the perceived emotion of the artist.

References

1. "Song Lyrics | Kaggle." *Kaggle, 2018, https://www.kaggle.com/paultimothymooney/poetry*
2. "danwaters/lfi: lyrics from image." *GitHub, 25 April 2021, https://github.com/danwaters/lfi*
3. "Text generation with an RNN | TensorFlow Core." *TensorFlow*, Google LLC, 02 April 2021, https://www.tensorflow.org/tutorials/text/text_generation
4. "NLP: Text Generation through Bidirectional LSTM model." *Towards Data Science, 30 Jan 2021, https://towardsdatascience.com/nlp-text-generation-through-bidirectional-lstm-model-9af29da4e520*
5. "Basic classification: Classify images of clothing." *TensorFlow, 19 March 2021, https://www.tensorflow.org/tutorials/keras/classification*

Appendix: Final Thoughts on CSCE 5218

I started this Master's journey with one goal in mind: really learn how to break down ML problems and get hands-on experience with TensorFlow. Through this project, I accomplished that, and have learned more about the practical side of machine learning than any class so far. Thank you for a great course.