Group 5 Final Report

Diego Llanes Andrew Holmes Avery Le Michelle Fast

Abstract

Task 1 aims to authenticate speaker identities using audio clips. We propose a model with an 8-layer transformer encoder generating rich embeddings to accurately represent speaker identities. This model places same-speaker embeddings closely and different-speaker embeddings far apart in the embedding space. Achieving an 80% accuracy rate, our model outperforms the 50% majority class prediction, showcasing its suitability for the task of voice recognition.

Task 3 aims to accurately convert graphemes into their corresponding phonemes. I propose the use of a sequence-to-sequence model utilizing a GRU-based encoder-decoder with an attention mechanism. The best results achieved was a validation phoneme error rate of 12%; achieved with a bi-directional encoder and auto-regressive decoder, sub-word tokenization, a positional encoding and token embedding. Along with other regularization techniques, the model was able to generalize to new data remarkably well.

Task 4 aims to distinguish between authentic and AI generated images of human faces. The proposed model was a 3 layer Convolutional Neural Network and was simplified to a simple Deep Neural Network with 2 Linear layers, which outperforms any model with convolutional layers. The most important finding was the use of error-level-analysis, which helped the model recognize features better by focusing on the error that AI images contain.

Task 5 aims to predict the starting day of a sequence of eight days given a global map of average spatial temperatures. The initial approach was 357-way classification but a shift was made to multivariate regression. The best performing model was, surprisingly, a single-layer DNN. The biggest breakthrough was mapping the days of the year to the unit circle and thus having the model predict the (x, y) coordinates instead of the "true" day.

I. Task 1: Speaker Verification

A. Methods

Data Preprocessing: For the final dataset, we create a dictionary mapping each Speaker to their corresponding audio files. We store the audio information in decibel-scaled Mel spectrograms because of their ability to capture rich representations of human speech, especially when logarithmically scaled (in decibels). We then apply random speed perturbations, temporal dropout, and frequency dropout [4].

Models: We tried a litany of model architectures, ranging from DNNs to transformers, but the architecture that netted the most gain was transformers. The main boost of performance that we received was changing the objective function of this task away from traditional classification and towards the cosine similarity of unitized embeddings.

Training and Tuning: The majority of the training was done on the CF-420 Lab machines due to cluster issues. The majority of the time this task was not spent on hyper-parameter tuning but rather architecture development; a brand new model would be built and run for some small period to see if it made meaningful improvements. We would run small sweeps to find sensible hyperparameters, just to ensure that we were giving each new architecture a fair chance.

Baselines: The baseline is majority class prediction, for this task we have artificially set the classes to be equal, in other words, we have a fifty-fifty split of the same speaker to different speakers. So majority class for this task is 50% accuracy, we believe that we have done much better than this because nearly all models reach $\sim 75\%$ accuracy.

B. Results

Figure one lists the architectures that were employed alongside their respective Dev Accuracies labeled as "Acc".

Architecture List	
Architecture	Acc
2 Layer Conv to Linear	50%
8 Layer Transformer Encoder to Linear	50%
Google Conformer [3] with Dot Prod	50%
2 Layer Conv with Dot Prod	70%
2 Layer Conv with Contrastive Loss	75%
8 Layer Transformer Encoder With Dot Prod	77%
3 Layer Conv to 8 Layer Transformer Encoder	70%
8 Layer Transformer Encoder With Contrastive Loss	80%

Fig. 1: Architecture List

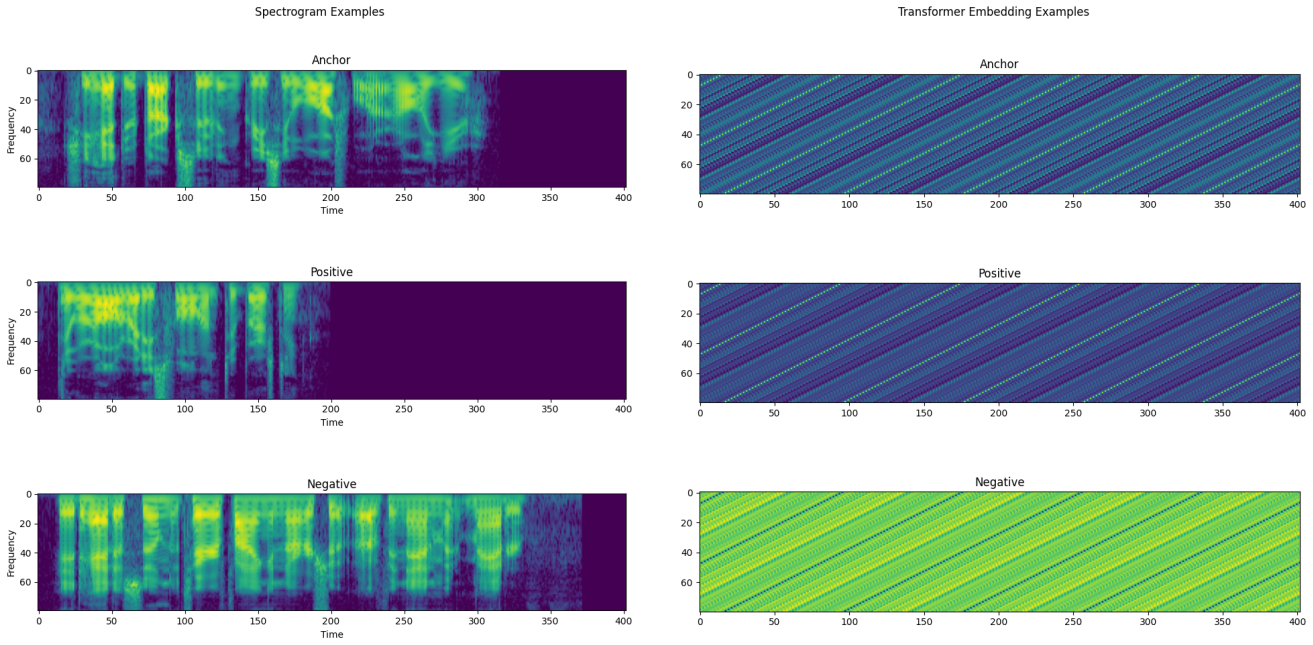
C. Conclusions

Through rigorous experimentation with various audio data, we gained valuable insights into the complexity of audio processing, necessitating the comprehension of techniques such as Short-Time Fourier Transforms, Spectrograms, the Mel Scale, and Decibel Scaling. Models output embeddings, which is some enriched, latent representation of the original input. Therefore, an optimal objective function would push embeddings of similar speakers closer and separate dissimilar ones rather than some arbitrary bucketing function.

We observed Convolutional Neural Networks adeptly capture temporal and spatial features in spectrograms. However, Transformers, due to their sequence processing capability, excel in audio tasks. The chosen model, an 8-Layer Transformer Encoder, optimizes the embedding task with a simple yet effective architecture. It processes three spectrogram inputs; an anchor, positive, and negative, aiming to align the anchor and positive closely in the embedding space while distancing the anchor from the negative. This approach exemplifies the superior performance of Transformers in audio processing tasks.

D. Contributions

Task 1 was led by Diego Llanes but all group members were there for the creation of the main pipeline and the original runner. We wanted all of our code to be as generalizable as possible, and at first it worked great but we inevitably started working on our own divergent branches as time went on.



(a) Mel Spectrograms **(b)** Generated Transformer Embeddings

Fig. 2: This demonstrates our model's capacity to push same speaker examples to the same embedding space through contrastive learning via Triplet Loss [8].

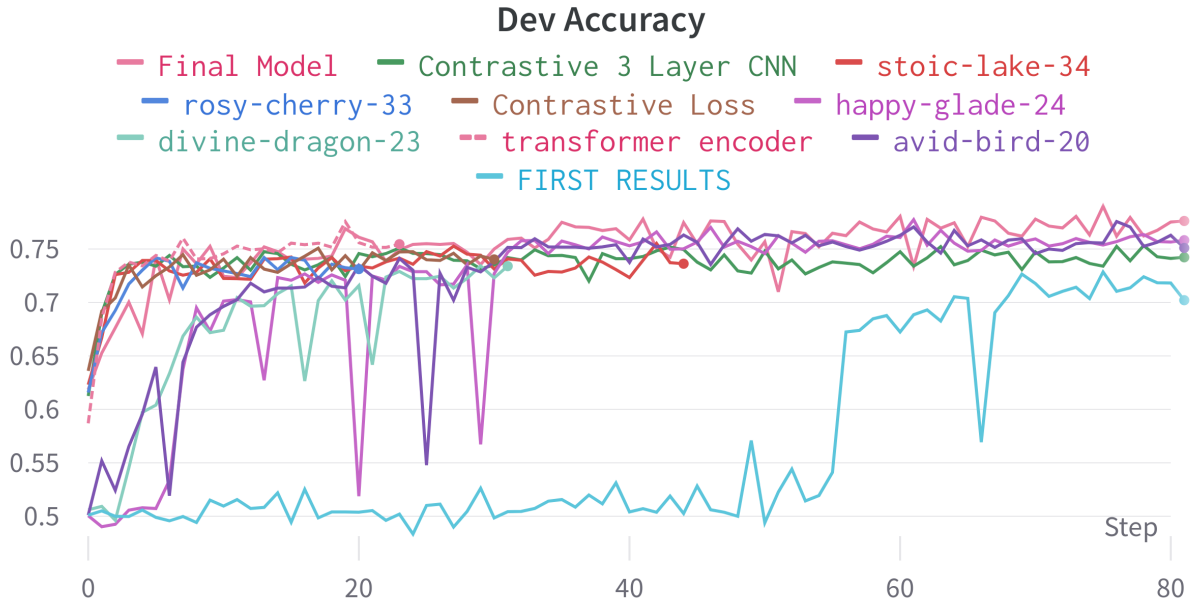


Fig. 3: This shows my Dev accuracy across different architectures (excluding ones that never learned). All of the lines that begin at $\approx 60\%$ are from using contrastive loss as the objective function, with everything below just using unitized dot-prod. This figure also shows the closest thing to Grokking [7] on non-algorithmic datasets, that we have ever seen.

II. Task 3: Grapheme to Phoneme

Overview: Grapheme to Phoneme conversion involves transforming written graphemes into their corresponding spoken pronunciations, represented by phonemes (e.g. the 'A' in 'CAT' $\rightarrow$ 'AE'). This task is usually done with a sequence to sequence model consisting of an encoder and a decoder.

A. Methods

Data Preprocessing: The data preprocessing is a crucial step for grapheme to phoneme conversion. For my approach, I perform sub-word tokenization, splitting the word into common trigraphs ('TCH', 'TIO', etc.) and digraphs ('SH', 'Th', etc.) before splitting on a character-level. By adding sub-word tokenization, the model is better able to learn common groupings of characters that can affect the phoneme sequence.

Models: Encoder-Decoder models have shown state of the art performance on machine translation tasks, as such, I tried many encoder-decoder models; vanilla RNN's, GRU LSTM's, and Transformers. A sequence to sequence model consisting of encoder-decoder GRU's yielded me the best results. The biggest increases in performance came from adding in bi-directional GRU's in the encoder and decoder, training on KL divergence loss, adding skip connections to allow gradients to flow easier, and adding in more regularization to avoid overfitting.

Training and Tuning: I added in $< s/ >$, $< e/ >$, and $< pad >$ tokens to the existing vocabulary. There are 67 grapheme tokens and 72 phoneme tokens.

After tokenizing the input and target, each sequence is converted into a token embedding with a learned positional encoding. The decoder had a 75% chance of teacher forcing the next token and there was a 25% chance of using trigraph or digraph representations. The models were trained using the AdamW optimizer and loss was computed by taking the KL Divergence Loss between predictions and targets, which had small amounts of label smoothing applied. The hyperparameters were tuned properly by finding the optimal nuisance hyperparameters first and then optimizing each scientific hyperparameter.

Model Type	Train PER (%)	Dev PER(%)
RNN	93.03	189.50
GRU	34.33	90.89
Bi-directional GRU	5.13	12.98
Ensemble	**5.12**	**9.89**
Transformer	3.67	89.76

Fig. 4: Results from best model of each type

B. Results

Metric: I used the common evaluation metric of Phoneme Error Rate, calculated by the number of insertions, deletions, and substitutions divided by the number of tokens in the target sequence.

Baseline: My baseline was a vanilla RNN encoder-decoder which lacked the capabilities of learning any meaningful patterns from the training dataset. The transformer model was too expressive and was prone to overfit to the train data. Even after heavy regularization and a decrease in the model complexity, the transformer was still unable to generalize.

Best Model: The best model I produced was an ensemble of three sequence to sequence models consisting of bi-directional GRU encoder-decoders. Each model has slightly different configurations and a different percent chance of teacher forcing and augmentation. Each model was trained individually and then the probabilities from each are averaged together to create an average prediction. This lead to a lower PER than each of the three models.

C. Conclusions

The baseline RNN seemed to lack all capabilities of learning as all of the parameter's gradients immediately zeroed out. After adding in skip connections, lowering the learning rate, and adding in regularization a more complex GRU model was able to learn a general trend in the data. It wasn't until the addition of sub-word tokenization, label smoothing, and positional encoders that the model was able to more accurately convert graphemes to phonemes.

D. Contributions

Task 3 was led by Andrew Holmes with contributions from all other members in the form of debugging and moral support.

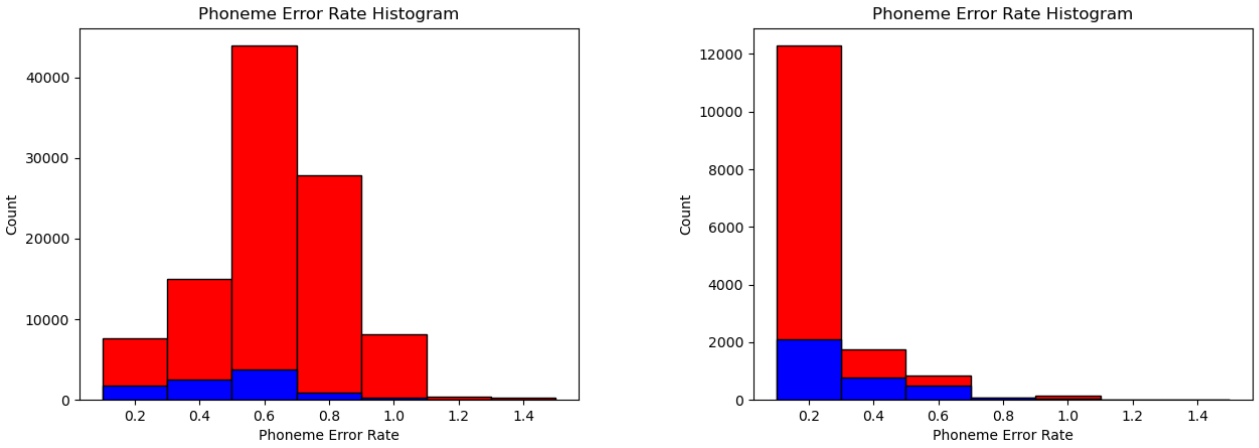
Grapheme	(True)	(Pred)
ANDREW	AE1 N D R UW0	AE1 N D R UW0
DIEGO	D IY0 EY1 G OW0	D IY0 EY1 G OW0
MICHELLE	M IH0 SH EH1 L	M IH0 SH EH1 L
AVERY	EY1 V ER0 IY0	EY1 V ER0 IY0

(a) Predictions from best model

Grapheme	(True)	(Pred)
ANDREW	AE1 N D R UW0	AE1 N D R AH0
DIEGO	D IY0 EY1 G OW0	D IY1 EH0 G 0W2
MICHELLE	M IH0 SH EH1 L	M IH0 SH AH1 L
AVERY	EY1 V ER0 IY0	AE1 V ER1 IY1

(b) Predictions positional encodings or bi-directional GRU's

Fig. 5: Qualitative performance analysis



(a) Distribution of dev PER at epoch 0

(b) Distribution of dev PER at epoch 19

Fig. 6: Histogram plots of dev per on train (red) and dev (blue) at beginning of training and at end of training

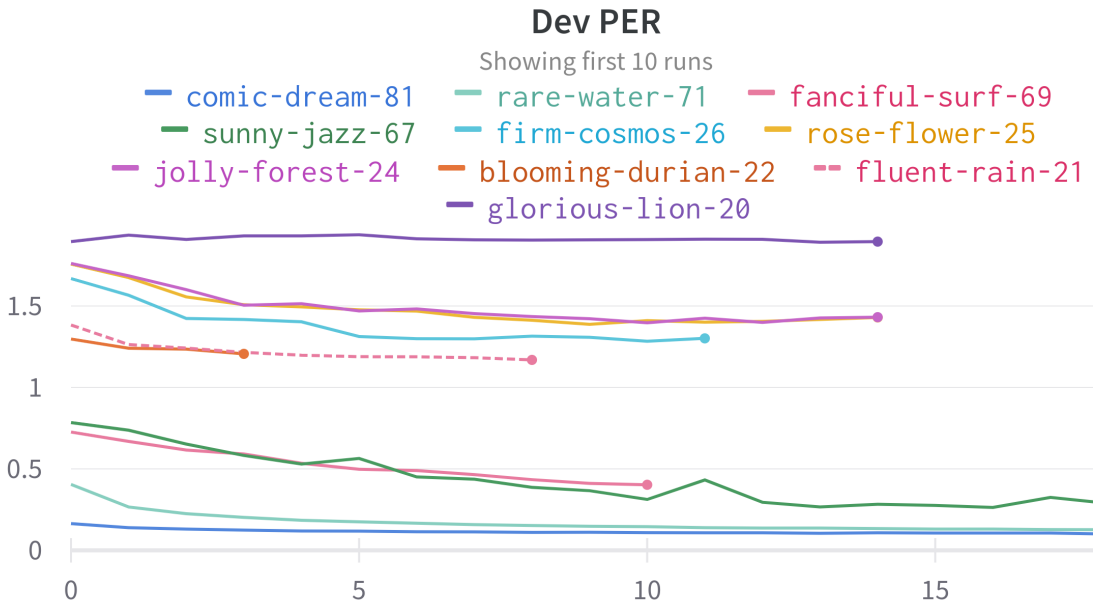


Fig. 7: Dev PER plot shows baseline (top), GRU (middle cluster), and bi-directional GRU's (bottom cluster)

III. Task 4: Fake Image Detector

A. Methods

Data Preprocessing: Standard image transformations were applied on both the training and validation datasets. The transformations consist of random cropping, resizing to 224×224 images, converting to PyTorch Tensors, and error level analysis (ELA). ELA is a popular preprocessing method in image forensics to calculate the absolute difference between an original image and its lower compression (90%) version, resulting in edge highlights which assist feature extractions and improve training [6].

Models Developed: The baseline model follows the state-of-the-art Convolutional Neural Network for image recognition, consisting of three layers (3-CNN), each followed by Batch Norm, ReLU activation, dropout, and max pool. The last layer goes through adaptive max pooling instead of regular max pooling. These convolutional blocks were followed by two fully connected layers with a ReLU activation in between.

When removing all convolutional layers to experiment with a Deep Neural Network (DNN), the model surprisingly excelled and is chosen for submission.

Additional models with one and two convolutional layers (1CNN and 2CNN) were trained to determine the importance of having convolutions.

Training and Tuning: Before the DNN was developed, 3CNN was continuously tuned using the same optimizer Adam and learning rate scheduler ReduceLROnPlateau to determine the best configuration for nuisance hyperparameters. Using Cross Entropy Loss, the resulting configuration produced the best performance for the baseline model:

- output channels: $8, 16, 16$
- kernel size: 3
- output features, adaptive max pooling size: 128
- activation function: ReLU
- dropout: $p = 0.25$ at each convolutional block and $p = 0.5$ at last convolutional block.
- learning rate: 0.0001

The DNN was trained using the same optimizer and learning rate scheduler. Further tuning resulted in training instability or over-expressiveness, therefore the optimal configuration was the same as 3CNN without the convolutional hyperparameters.

Baselines: The baseline 3CNN model was consistently refined overtime, reaching an average of 98% in validation accuracy after ~ 150 epochs, while the DNN model converged much faster, averaging 99% at ~ 30 epochs.

B. Results

For better interpretation, accuracy scores are visualized for all experiments. Learning rate proved to be a primal factor, as the accuracy greatly fluctuates between epochs when it was 0.001 as opposed to 0.0001. Adding ELA as a transformation significantly enhanced the accuracy and established a foundation for subsequent experiments, as visualized in figure 8. It also helped in clarifying facial features such as the face outline, shadow, background, etc., which directly contributes to the predictions. The saliency map (fig. 11) demonstrates what features the model focus on and how the gradients interact.

The study of models revealed no discernible relationship between the number of convolutional blocks and performance, as the 2CNN performed worst while the 1CNN was only suboptimal to the DNN (fig. 9,12). Different acivations on the DNN were also interesting, as Swish helped the model learn faster but produced unstable loss over training (fig. 10), while ReLU promoted consistent, gradual learning, both in loss and accuracy scores.

C. Conclusions

One of the most surprising findings is the use of convolutional layers, which seems to be most effective when only one block is utilized. Through various experiments with data preprocessing and hyperparameter tuning, the optimal model was found to perform best as a double layer Deep Neural Network with exhaustively preprocessed data as well as ReLU activation function and a reasonable number of output features.

D. Contributions

Task 4 was led by Avery Le, with the assistance of Diego Llanes in compiling the backbone configurations for a unified experiment process, Michelle Fast in brainstorming relevant studies to be experimented with, and Andrew Holmes in initializing WandB for visualizations.

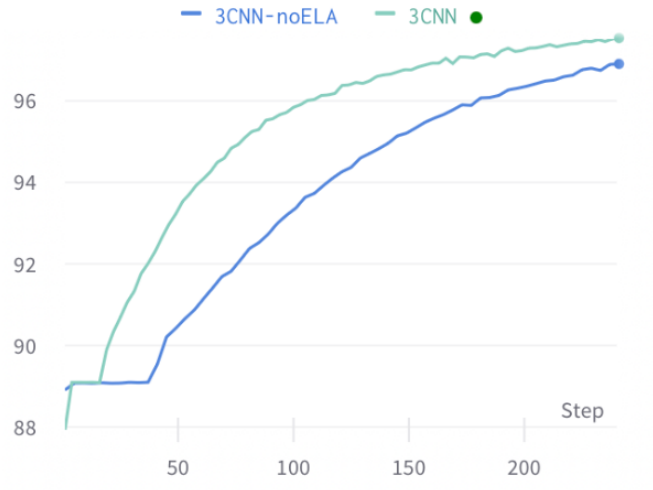


Fig. 8: Adding ELA improved accuracy and promoted better learning

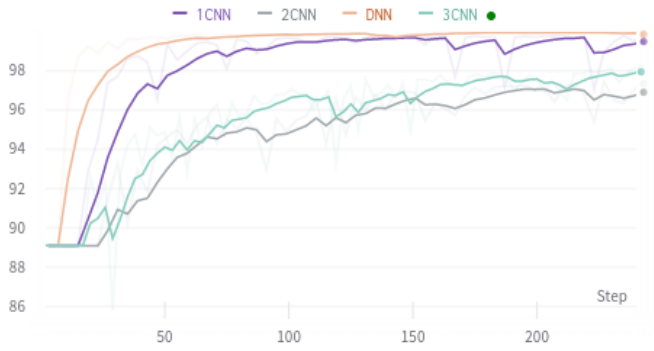


Fig. 9: All models' performance over 240 epochs

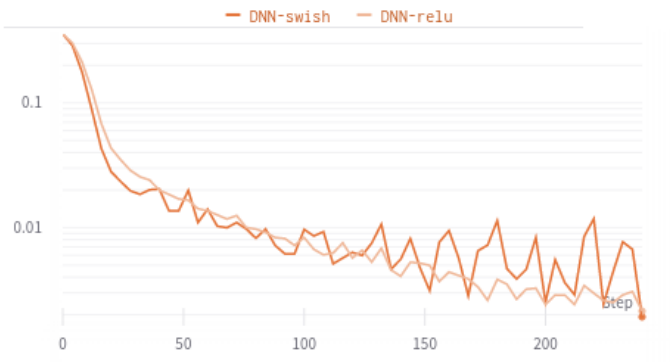


Fig. 10: Loss of different activation functions on the DNN, $log(y)$ scale

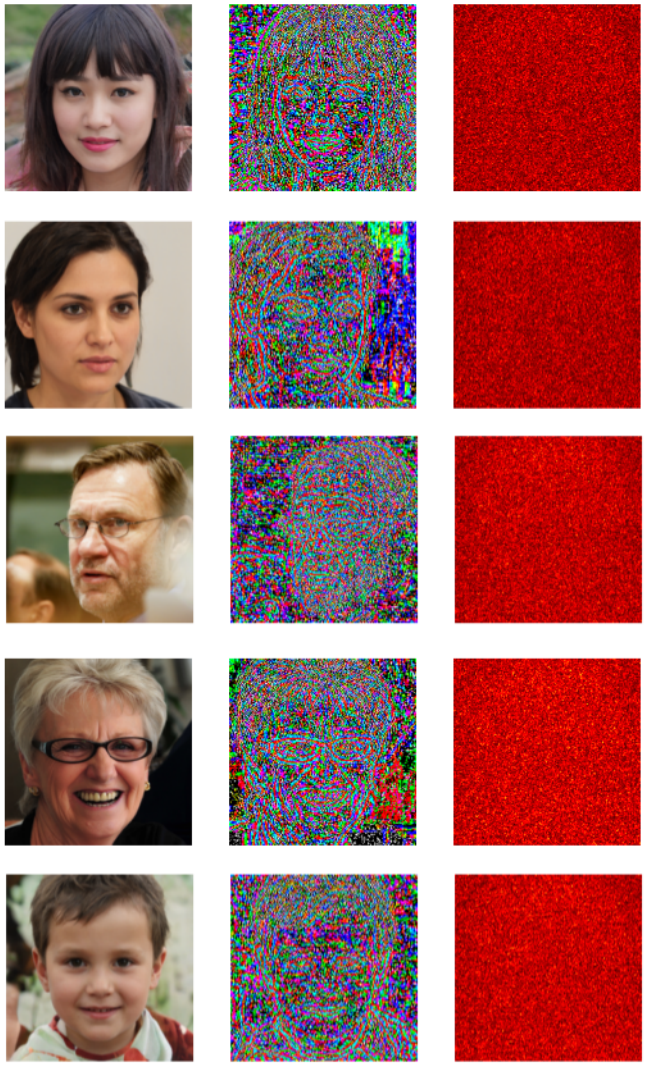


Fig. 11: From left to right: original image, transformed image, saliency map

Model	T.Loss	D.Loss	T.Acc(%)	D.Acc(%)
DNN	**2.14e-3**	**9.31e-4**	**99.93**	**99.98**
1CNN	1.37e-2	1e-2	99.68	99.81
3CNN	4.89e-2	4.35e-2	98.44	98.67
2CNN	7.4e-2	6.42e-2	97.26	97.77

Fig. 12: Breakdown of each model's best performance

IV. Task 5: Localizing Weather in Time

A. Methods

Data Preprocessing & Feature Extraction: Instead of predicting the "true" days, ranging from 0 to 356, the data was preprocessed using cyclic encodings to map to coordinates on the unit circle. This added cyclic time qualities to the model's inductive bias. To transform between the above modes, Listings 1 and 2 were used. Along with this change, the data was normalized using the mean and standard deviation of the training dataset.

Models & Baselines: As a baseline, 357-way classification was attempted on a single hidden-layer (SHL) DNN, multi-layer (ML) DNN, and CNN. Each of these models predicted the same day every time, and reached a Cross-Entropy loss of 5.86 at best (Figure 13a).

The next family of attempted models performed multivariate linear regression and predicted cyclic encodings – unit circle (x, y) coordinates – instead of the true days. An SHL DNN implementing LayerNorm, ReLU, and an output activation of tanh proved to be the most successful model, though several more variations of this were tested and analyzed (Figure 13a).

B. Training & Tuning

Learning rate, training data, and activation choice all impacted model performance significantly. Low learning rate used in conjunction with ReduceLROnPlateau scheduler allowed the model to converge into a true minima instead of skipping over it. Prior to training on the cluster, the model was trained on a seventh of the available training data. Once trained on the cluster with all available training data, this same model achieved a more impressive MSE and better generalization (see Figure 13b). Finally, comparing runs between LayerNorm, ReLU, Swish, and tanh activations allowed for significant optimization.

C. Results

Performance of the classification models (Figure 13a) made it apparent that classification was not the approach to take, and that data preprocessing was vital. Although tough to quantitatively compare the classification and regression models (Figure 13b), classification models qualitatively performed much worse, likely due to the lack of normalization and data preprocessing. The regression models, attaining these qualities, performed much better even when trained on a seventh of the data. The "best" model was determined by comparing performance on this small data for ease of computation, then was ran on the cluster with all of the training data. The final model, a SHL regression DNN with LayerNorm, ReLU, and a tanh output activation achieved a train and dev MSE of 2.71e-4 and 2.96e-4, respectively.

D. Conclusions

Throughout the experiments outlined above, the most interesting findings were using LayerNorm as an activation function and cyclic encodings. An attempt at a simple regression model utilized LayerNorm as an activation function, and reached impressively good results (Figure 13b). Unknown prior to further research, LayerNorm can be used as a general-purpose activation function [5]. The biggest breakthrough was the use of cyclic encodings, which the model with an observable relationship between day 356 and 0 and forced the model to predict between -1.0 and 1.0, minimizing the variance between predictions.

The best results came from a SHL DNN performing multivariate linear regression with LayerNorm & ReLU between the layers and tanh as an output activation. LayerNorm provides deterministic normalization, which proved to be very useful in this task. As for activation functions, it would have been expected for Swish to outperform ReLU because of its trainable parameter, but Swish has been proven to only outperform ReLU on networks with more than 40 layers [1,2]. The decision to include tanh as an output activation forced the model into predicting numbers between -1.0 and 1.0 in order to align with the unit circle. A learning rate of 0.0001 was chosen after comparing the performance of several models with learning rates between 0.001 and 0.00001. Performance on the dev set is visualized in Figure 14.

E. Contributions

Diego Llanes and Andrew Holmes contributed to brainstorming about the architecture of the model, as well as analyzing LayerNorm's effects. Avery Le assisted in running experiments on the cluster.

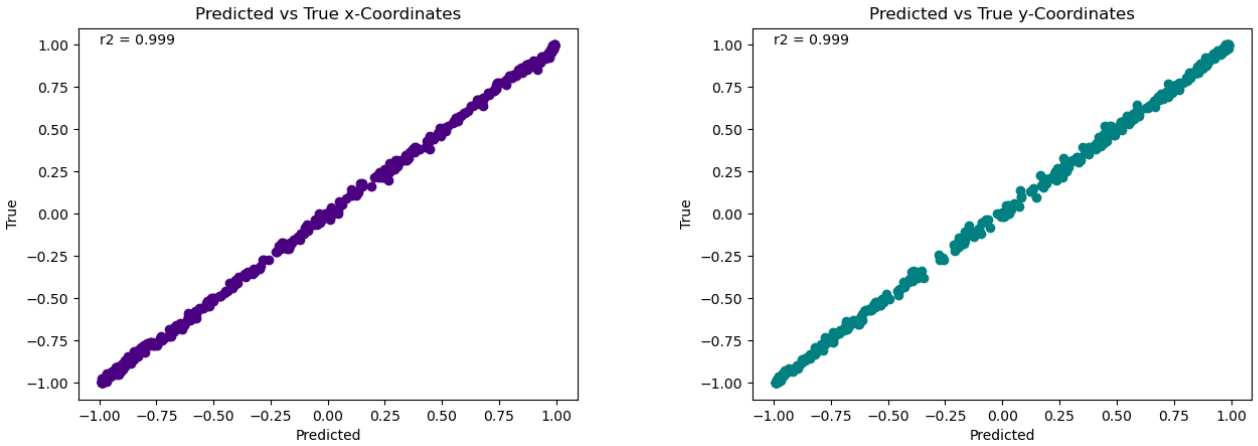
Model	Training Loss	Dev Loss
SHL DNN	5.91	5.93
ML DNN	5.91	5.92
CNN	5.86	5.87

(a) Classification Models (Cross-Entropy)

Model	Training Loss	Dev Loss
Linear Regression	3.1e-1	3.4e-1
LayerNorm (LN)	2.5e-3	2.6e-3
LN, Swish	7.84e-4	8.77e-4
LN, ReLU, $1/7$ Data	6.21e-4	8.88e-4
LN, ReLU, Cluster	2.71e-4	2.96e-4

(b) DNN Regression Models (MSE)

Fig. 13: Performance of Classification and Regression Models



(a) x-Coordinates

(b) y-Coordinates

Fig. 14: One-to-one plots of predicted vs true x and y coordinates.

```python
def day_to_coord(day):
    # Unit Circle encoding of day
    angle = (2*math.pi) * (day - 1) / 357
    x = math.cos(angle)
    y = math.sin(angle)
    coord_list = [x,y]
    unit_coord = torch.Tensor(coord_list)
    return unit_coord
```

Listing 1: Day to (x, y) Coordinates

```python
def coord_to_day(cos_sin):
  # Get individual coordinates
  cos = cos_sin[0].item()
  sin = cos_sin[1].item()
  angle = math.atan2(sin, cos)

  # Account for cyclic properties
  if angle < 0: angle += (2 * math.pi)

  # Calculate the day
  cont_day = angle * 357 / (2 * math.pi)
  day = int(cont_day) + 1
  return day
```

Listing 2: (x, y) Coordinates to Day

V. REFERENCES

[1] Swish Activation Function by Google (Medium Description). https://medium.com/@neuralnets/swish-activation-function-by-google-53e1ea86f820.

[2] Searching for Activation Functions (Archival Paper). https://arxiv.org/pdf/1710.05941.pdf

[3] Conformer: Convolution-augmented Transformer for Speech Recognition. https://arxiv.org/pdf/2005.08100.pdf

[4] SpecAugment: A New Data Augmentation Method for Automatic Speech Recognition. https://ai.googleblog.com/2019/04/specaugment-new-data-augmentation.html

[5] On Layer Normalization in the Transformer Architecture.https://arxiv.org/pdf/2002.04745.pdf.

[6] Detection of Tool Based Edited Images From Error Level Analysis and Convolutional Neural Network. https://arxiv.org/pdf/2204.09075.pdf

[7] Grokking: Generalization Beyond Overfitting On Small Algorithmic Datasets. https://arxiv.org/pdf/2201.02177.pdf

[8] FaceNet: A Unified Embedding for Face Recognition and Clustering https://arxiv.org/pdf/1503.03832.pdf