Practical - CS181

Kushal Chattopadhyay, Sid Bharthulwar, Mayesha Soshi
kchattopadhyay, sbharthulwar, maysoshi

April 6, 2023

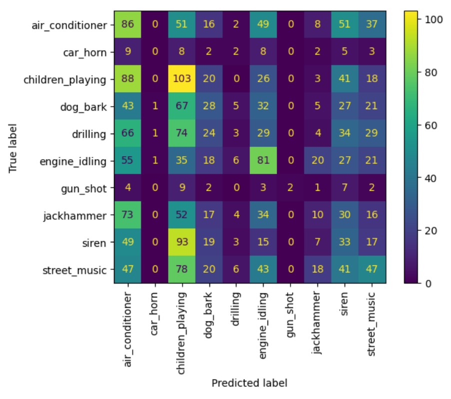
1 Part A: Feature Engineering, Baseline Models

1.1 Approach

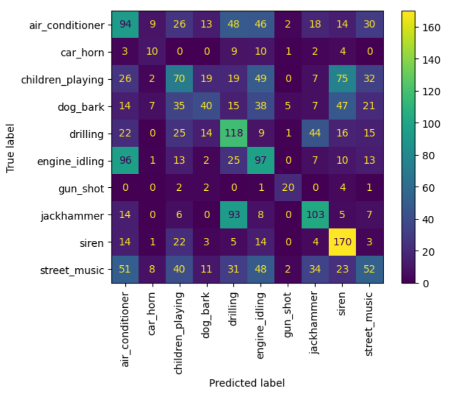
To find our first baseline, we trained a logistic regression model on the raw amplitude features and processed Mel spectrograms. We implemented a L2 regularization to combat overfitting in the regression model. We set our tolerance to the default value of 1e-4, meaning that the optimization process will stop when the change in the objective function or the parameter vector is less than 0.0001. We additionally set the iterations to 1000 as we noted that the regression converged with that many iterations and set tolerance value. We did not utilize PCA for our logistic regression method so training was computationally intensive (up to 11,000 dimensions for a single point). However, applying PCA would have allowed us to reduce the features of our data points.

1.2 Results

We provide the train, test, and class 0 (air conditioner) accuracies, as well as confusion matrices for our logistic regression to better understand where the regression performs well.



(a) Baseline 1



(b) Baseline 2

Figure 1: Baselines 1 and 2 (Logistic Regression) Confusion Matrices

Model	Train Acc (%)	Test Acc (%)	Class 0 Acc
Logistic, amp	0.9773	0.1788	0.1654
Logistic, mel	0.9827	0.3523	0.3133
KNN, k=5	0.5953	0.2690	0.1667
KNN, Tuned	0.7435	0.2599	0.2200
Decision Tree	0.4612	0.1972	0.1867
Random Forest	0.79236	0.2216	0.0533
CNN	0.9498	0.519	0.3005

Figure 2: Train accuracy, test accuracy, Class 0 (`air_conditioner`) accuracy

Label	**Baseline 1**	**Baseline 2**
air_conditioner	0.286667	0.313333
car_horn	0.000000	0.333333
children_playing	0.344482	0.230769
dog_bark	0.122271	0.174672
drilling	0.011364	0.450758
engine_idling	0.306818	0.382576
gun_shot	0.066667	0.666667
jackhammer	0.042373	0.457627
siren	0.139831	0.733051
street_music	0.156667	0.176667

1.3 Discussion

We notice that our Mel spectrogram data performs much better on test data than the raw amplitude data whereas they both have very high training accuracies. Across all labels, the Mel spectrogram data on average performs better per class than the raw amplitude data, with some extremely high accuracies in the Mel spectrogram on labels that we often note low accuracies for from raw amplitude data. These include `siren`, `gun_shot`, and `jackhammer`. I hypothesize that this is because the Mel spectrogram data provides information on audio-related features in each time window, whereas the raw sound data solely provides the amplitudes. It can be inferred that although amplitude patterns can correspond to multiple potential classes, audio-related features encoded in spectrograms are likely more unique to certain classes. Thus, the test accuracy would be improved. Additionally, the raw sound data may overfit more to the amplitudes than the Mel spectrogram data as the latter is more informative on the features in each time window that demarcate a sound as being from a certain class.

Our intuition for why the baselines are so low are due to the limitations of linearized classification techniques such as logistic regression. Contemporary neural network architectures, such as LSTMs, Transformers, RNNs, and CNNs, are specialized for different data modalities. CNNs perform well on images/videos, whereas LSTMs, Transformers, and RNNs perform well on sequential data such as text and time series. Due to the Universal Approximation Theorem, neural networks can segment the latent space of the data more finely than linear models can, so our baseline models perform comparatively worse to neural networks.

2 Part B: More Modeling

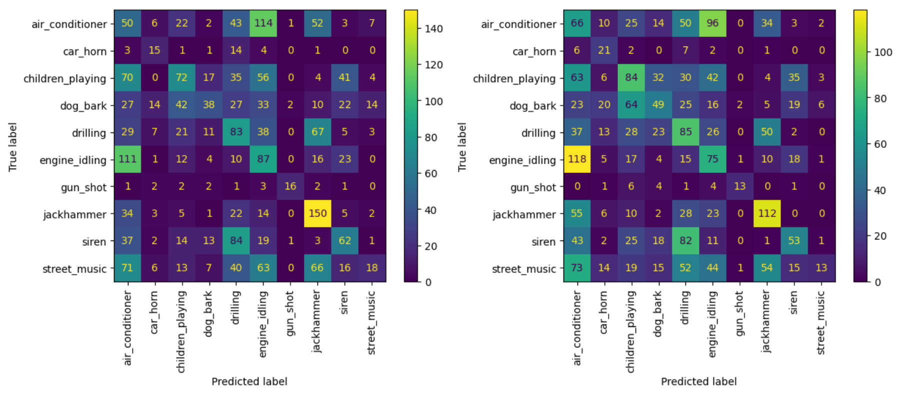
2.1 First Step

2.1.1 Approach

We ran a KNN Classification on our Mel spectrogram data with a default k value of 5. This means that for each data point in the spectogram dataset, the algorithm identified the 5 nearest neighbors and classified the point based on the majority classification of the 5 neighbors. We intuitively believed that setting a k value of 5 would allow us to better understand how the KNN performs on the data and how we should adjust our k to better reflect the data in following iterations. This is done using the default kernel as we were unsure of the relationships of the dataset, although we would perhaps be able to improve our performance by using other kernel types.

2.1.2 Results

The corresponding confusion matrix (including the hyperparameter-tuned model) and per-class classification accuracy of the KNN model is included below. The overall train and test accuracy was **0.595** and **0.269**, respectively.



(a) Default (b) Grid Search

Figure 3: KNN Classification Confusion Matrices

Label	KNN
air_conditioner	0.1667
car_horn	0.3846
children_playing	0.2408
dog_bark	0.1659
drilling	0.3144
engine_idling	0.3295
gun_shot	0.5333
jackhammer	0.6355
siren	0.2627
street_music	0.0600

2.1.3 Discussion

Compared to the Mel spectogram logistic regression model from Part A, the KNN Classification performed significantly worse. In Part A, Baseline 2 had a train accuracy of 0.983 and a test accuracy of 0.352 whereas KNN Classification with $k = 5$ has a train accuracy of 0.595 and a test accuracy of 0.269. The lower train accuracy suggests that the model is not learning well from the data which would lead to poor performance on both the train and test sets. This may be a result of the k value – considering the size of our dataset, the default k value of 5 is definitely too small. It is difficult to intuitively select a k value that would improve the performance of the model because we do not know how the feature parameters and the classification values are related. There also is class imbalance which could lead to inaccurate classification for less represented classes. Thus, we use GridSearch in the next section to improve the performance of our model.

2.2 Hyperparameter Tuning and Validation

2.2.1 Approach

For hyperparameter tuning, we used two models: **(1) KNN Classification** and **(2) Decision Tree**.

We tuned the hyperparameter of the KNN k by performing a grid search. To optimize this search, instead of iterating over every possible k (from 1 to the amount of data points), we instead only checked powers of 2. We then used this trained model with k chosen according to grid search to obtain our predictions and train / test accuracy. As k increases, test accuracy increases (overfit decreases); however, past a certain k value, too many points are being averaged at every iteration which leads to underfitting (i.e. the average model). Thus, it is very important to tune the parameters of the model and identify the best k.

For the decision tree, we tuned the tree depth hyperparameter by iterating through a list of potential tree depths and training a decision tree model on it. We iterated over 10 values for this hyperparameter. We performed a 5-fold cross validation at each iteration and obtained the corresponding train and validation accuracies. A plot of the train and validation accuracies is included below for tree depths 1-10. As we can see, as the depth of the tree increased, the train accuracy increased significantly; this makes sense because as we increase the depth of the tree, the model is able to learn the patterns and parameters of the data better. However, increasing the tree depth too much can lead to overfitting in the training set and poor evaluation in the test set. We compared the simple train accuracy to the 5-fold cross validation accuracy to obtain the best tree depth for our model which was 9. We did not test for larger tree depth as we noticed that the validation score had plateaued; thus, we went with the best tree-depth value in the 1-10 range. We then used this tree depth to train a random forest model (with 50 decision trees) to obtain even more robust results.

2.2.2 Results

For our KNN model, we obtained a train accuracy of **0.744** and a test accuracy of **0.259**. For our hyperparameter tuned decsion tree, we obtained a train accuracy of **0.461** and a test accuracy of **0.198**. For the random forest model with depth 9 and 50 trees, we obtained a train accuracy of **0.782** and a test accuracy of **0.223**.

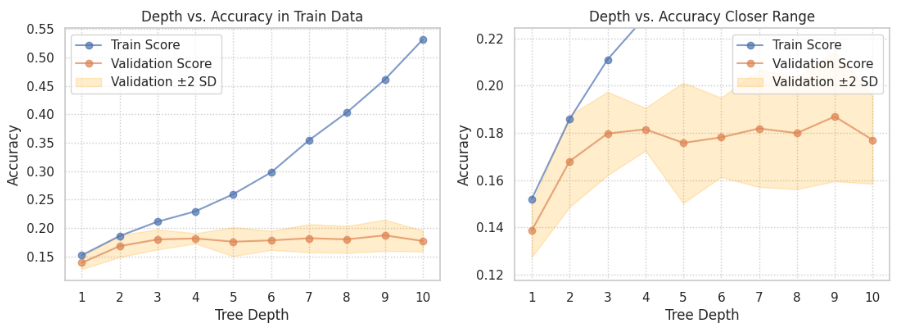
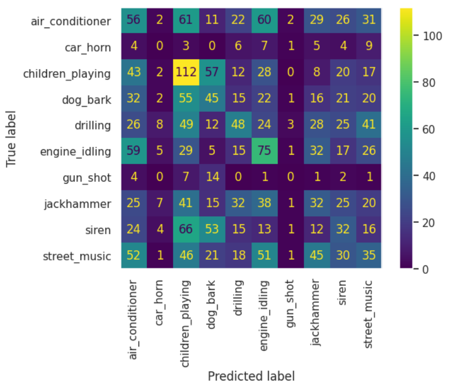
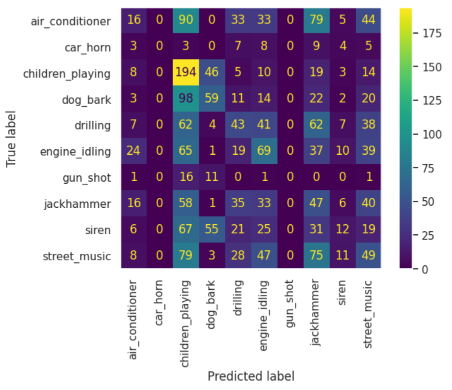


Figure 4: Hyperparameter Tuning for Decision Tree



(a) Decision Tree



(b) Random Forest

Figure 5: Tree Classification Confusion Matrices

Hyperparameter Tuned Model Accuracies

Label	**KNN Tuned**	**Decision Tree**	**Random Forest**
air_conditioner	0.22000	0.18666	0.05333
car_horn	0.53846	0.00000	0.00000
children_playing	0.28093	0.37458	0.64882
dog_bark	0.21397	0.19651	0.25764
drilling	0.32167	0.18182	0.16288
engine_idling	0.28409	0.28409	0.26136
gun_shot	0.43333	0.00000	0.00000
jackhammer	0.47457	0.13559	0.19915
siren	0.22457	0.13559	0.05084
street_music	0.04333	0.11667	0.16333

Model	Train Acc (%)	Test Acc (%)
KNN, Tuned	0.74356	0.25990
Decision Tree, AMP	0.46119	0.19799
Random Forest, AMP	0.79236	0.22166

2.2.3 Discussion

The hyperparameters of models can ultimately determine how well a model fits a dataset – with the KNN model, a small k value may lead to overfitting due to not taking into consideration other data points, whereas a large k value may result in a more averaged model that does not capture the relationships of the data. As a result, we expect a default value of k not to always perform the best on all datasets, as we see with our dataset, where the small default k value leads to poor train data. However, after iterating over several possible k values, we are able to improve our accuracies by ensuring that the KNN both incorporates variation in the data while not overfitting to the training set. The same logic applies to other nonlinear models – with decision trees, choosing an arbitrary depth may either overfit to training data or not have sufficient depth to be able to represent the relationships within the data. As a result, tuning the hyperparameter prior to using the model allows us to obtain an improved model. We iterated over multiple depths as our chosen hyperparameter and noted that performance on train data increases steadily with increasing depths, whereas test data performance quickly stagnates. We validated this by running our model on the depths and k values with optimal performance on test data.

Indeed, both our decision tree and random forest had higher test accuracy for the amp data, showing that hyperparameter tuning does increase accuracy and performance of our models. The decision tree was only slightly better than the baseline (with an increase of roughly 2%) whereas the random forest achieved 22% accuracy, which is most likely a product of the aggregation of the 50 decision trees. For such a complex model, the decision tree and random forest did not perform nearly as well as expected. The performance can be improved by using more robust hyperparameter tuning, such as experimenting with different splitting criteria or feature selection methods. We can also consider increasing the number of trees for the random forest to allow for more aggregated data.

The KNN, on the other hand, achieved a 25% test accuracy post-hypertuning which is slightly less than the default k-value of 5. The reason behind this could be the train accuracies of the two KNN models. The train accuracy for KNN model with $k = 5$ was 0.5953 whereas the hypertuned KNN model has a train accuracy of 0.7435. Evidently, the hypertuned KNN model is fitting to the training data better which could indicate some overfitting and more generalization to the test data. Regardless, both of the KNN models performed relatively well on the test set and can be used as an reliable model for classification.

3 Final Write-up and Reflections

3.1 Discussion:

Our ultimate sequential process was as follows. First, we trained the two baseline models (logistic regression) on the amp and mel data, respectively. Then, following the directions of the practical, we moved onto non-probabilistic models (decision trees, random forests, KNNs) for classifying the sounds. Finally, we realized during the data preprocessing stage that the mel spectrogram data was represented in 2D, and thus a CNN could perform well. Each step of the process provided us with valuable insight that carried over to the next step. For example, while wrangling with the high-dimensional mel spectrogram data while making the KNN classifier, we realized that the 2D data was perfect as input for a CNN-type model, so that became our logical next step of exploration. Additionally, different adaptations within the models improved accuracies as well. For instance, modifying the CNN architecture and repeatedly running experiments proved to be valuable towards determining a final CNN architecture with the best possible performance. This practice of learning from prior results was helpful, as our final CNN architecture performed approximately 4 times better than our original logistic regression baselines.

However, there are still some natural avenues of future exploration to consider. For example, utilizing k-fold cross validation could help increase confidence in the results of the model. Since we have an arbitrary train/test split, where data distributions are not guaranteed to be approximately equivalent, our model could be performing far better on train data than test data because of differences in the distributions of classes. Running k-fold cross validation across a unified dataset would more uniformly estimate the predictive capabilities of our model. Secondly, experimenting with more advanced model architectures could increase performance significantly. For example, note that the CNN (a far more complex model than logistic regression or KNN) performed better than any rudimentary model. For the amplitude data, which represents a time series, a model such as a recurrent neural network, transformer, or LSTM could perform better, since the locality of data points in relation to other data points is considered. Otherwise, the data is very high-dimensional, since each temporal measurement is considered a new dimension for the point. Finally, data augmentation would be an obvious next step when considering the real-world usability and impact of the classifiers we trained. Class imbalance within datasets isn't always expressed in the experimental accuracies of models. For instance, if 95% of a train/test dataset has 1 class and the other 5% is another class, always predicting the first class results in a 95% accurate model. However, in the real world, these tail-end 5% outcomes are important, so artificially augmenting the dataset to remove class imbalance could help the classifier detect rarer sounds, such as gunshots and car horns.

3.2 Reflections

3.2.1 1. Data Pipelines

We were provided with the Mel spectrograms, and did not invoke the code to transform the amplitude data to them. Instead, the only preprocessing we had to do was to one-hot encode the outputs for both baseline logistic regression models, as well as for the CNN. We utilized the raw amplitude data and Mel spectrograms as inputs to our models. The CNN stands out as most relevant as we are dealing with image data, and it also performs exceptionally well on test data.

3.2.2 2. Model Selection

Logistic regression as baselines were recommended by the practical instructions, and serve as some of the most basic multi-class classifier models available – however, it performs poorly in modeling nonlinear decision boundaries. Its simplicity makes it a good choice as a baseline. KNNs, decision trees, and random forests are the most popular and fundamental non-probabilistic models, and thus were optimal for the second part of the investigation, as they allowed us to utilize other representations of our data. Finally, as previously mentioned, CNNs perform well on multi-dimensional data (such as the Mel spectrograms).

3.2.3 3. Model Tuning

We used L2 regularization on the Logistic regression models to prevent overfitting on train data. On the CNN, we went through several iterations of model architecture to find the most performant model. For the decision tree, we iterated through different tree depths to identify the tree depth with the highest accuracy. We also did cross-validation and ensembling when doing our random forest model. Lastly, on the KNN classification, we used grid search to identify the best k-value.

3.2.4 4. Bias-Variance Tradeoff

We noted that there was a very considerable disparity in the train and test accuracies for every model. Across the board, this indicates a large degree of overfitting. On the logistic regression model, we used L2 regularization with a 1e-4 stopping criteria to prevent overfitting. On the CNN model, we defined explicit convergence criteria to prevent overfitting. On the random forest model, we aggregated 50 decision trees to reduce the variance of the individual decision trees; the bias was, however, unchanged since our tree depth remained the same. For the KNN, we did not implement additional methods to address the bias-variance trade-off. One way we can improve the KNN model is by either using a different distance formula or by iterating through all possible k values and selecting the most optimal one (since in our current implementation, we only checked powers of 2).

3.2.5 5. Evaluation

We considered the classification accuracy presented in the practical assignment documentation. However, as this value is simply a scalar for any model, we found confusion matrices for the 10 classes' prediction accuracies of each model to be a far more comprehensive set of metrics. These display the true positive/false positive/false negative values for each class (as well as which sounds incorrectly classified sounds are classified to, for each class), giving us a very substantial overview of the limitations of each model. In general, a good model constituted a confusion matrix where most values were consolidated in the diagonal of the matrix. For the KNN, we ran a grid search to implicitly determine the optimal k value, which had the advantage of allowing us to better model the data but the downside of taking a long time, even when only considering powers of 2. We considered modifying the kernel but figured that since we were unsure of the exact connections between data points, by Occam's razor we decided to go with the default. For the decision tree, we were able to plot the test and train accuracies over various depths and noted that the test accuracy plateaus after depth 2 while training data increases as depth increases, as to be expected as the tree overfits with more depth. Choosing depth as a metric allowed us to simply quantify overfit and underfit.

3.2.6 6. Domain-specific Evaluation

While none of us are familiar with the domain of audiology and signal processing for audiological applications, some domain-specific intuitions explained our results intuitively. For instance, the low classification accuracies on the gun shots and car horns classes were due to low representation of these two sounds in the dataset. This was likely due to the relative rarity of these types of sounds in New York City, compared to other sounds. Additionally, it makes sense that street music was commonly mistaken for children playing and engines idling were commonly mistaken for air conditioners. Our models make intuitive sense as they are classification models, and especially since we are dealing with image spectrogram data, the CNN performed well. Additionally, classes with more representation had higher accuracies in the models which makes sense since there was more data and thus, the model was able to learn the characteristics of the class better. Similarly, classes with low representation has low accuracies since the model was not able to learn about its characteristics well.

3.2.7 7. Design Review

In retrospect, we could have chosen more different model types to test our dataset on. For instance, LSTMs and RNNs perform well on time series data. Training and validating these model classes on the amplitude data could demonstrate far superior performance compared to our baselines. Additionally, we could have implemented data augmentation to alleviate the class imbalance issues that plagued our train and test datasets, leading to higher classification accuracies for underrepresented classes such as gunshot sounds. Some issues we expect in the future include that our models can often overfit, and large datasets like this one result in slow runtimes and scalability being impractical. To optimize, we could limit our search for k even further or perhaps iterate over less depths for our decision tree. And, of course, implement PCA to reduce the dimensionality of our data.

3.2.8 8. Execution and Implementation

Considering that we are classifying ordinary sounds, there are not many ethical considerations here. However, that being said, there are many real-life applications of a classification model like this. Classifications models can be used in a multitude of criminal (and other) cases to help identify different sounds. If we have videos or recording of different suspects, classification of the sounds can help us narrow down neighborhoods and locations of the suspect. Classification models can also be used in automotive vehicles to help identify warning signals from other cars, such as sirens and emergency vehicles. That being said, our current model is not robust enough for real-life deployment. We need to gather more data and allow our models to learn more to obtain a model that could be deployed in real life. We especially need to obtain more data for the less represented classes, such as `car_horn` and `gun_shot`.

4 Optional Exploration, Part C: Explore some more!

4.1 Approach

The Mel spectrogram data was two-dimensional, and we realized that convolutional neural networks(CNNs), a type of neural network architecture used in image/video tasks, are very performant in this case. No preprocessing besides one-hot encoding the outputs is required here as the Mel spectrograms are already represented as 2D images.

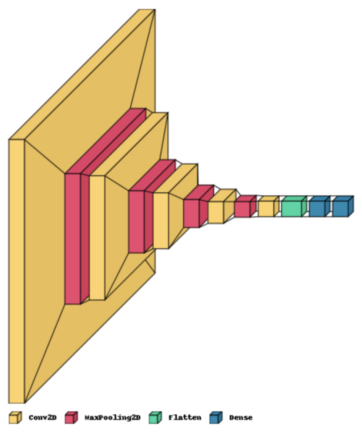


Figure 6: Architecture of the CNN

Picture above is the architecture for the CNN. Note the `conv2d` and `maxpooling2d` layers, which essentially downsample the image to a lower dimensionality. With ReLU activation functions and backpropagation to update all weights/biases, the CNN learns the most prominent 2D visual features and preserves them in the smaller latent representation. Note that the 3rd dimension of the latent tensor becomes 64, since we apply the convolutions with 32 followed by 64 filters. After a succession of these layers, we flatten the $28 \times 18 \times 64$ tensor into a 1×32256 vector. Now, we apply traditional dense layers on this vector (as showed in class) to further reduce the vector down to 1×10–the same size as our outputs. Finally, we apply the softmax activation function to bound all predicted class probabilities to between 0 and 1. Note that there are over 3 million parameters for the first dense layer. We originally thought that this would be an issue, since more parameters generally increases the likelihood for overfitting. However, after experimentation with different architectures, we found that architecture semantics didn't matter.

4.2 Results

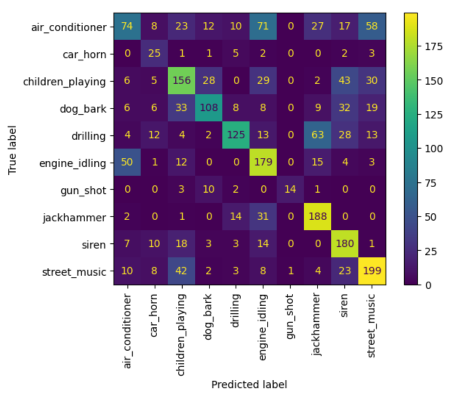


Figure 7: 10-Class Confusion Matrix for CNN

The CNN is trained for 15 epochs with the SGD optimizer. Past 15 epochs, our model began to overfit to the training data. Our final **train accuracy** was **0.950** and the final **test accuracy** was **0.519**. Below is the confusion matrix evaluated on the test data for the CNN we trained. Note that the diagonal of this matrix is comparatively stronger here compared to on our prior models, indicating a better model. However, note the low numbers of correct predictions for `car_horn` and `gun_shot` – a result of class imbalance (both `car_horn` and `gun_shot` each represent less than 4% of the dataset). Other results also intuitively make sense. For example, many `jackhammer` noises were misclassified as `drilling`, which makes sense because these sounds sound alike. Overall, the significantly stronger performance of the CNN compared to every other model type confirms our prediction that CNNs are comparatively more performant on two-dimensional data modalities. It also confirms to us that, by the Universal Approximation Theorem, neural networks are capable of approximating more complex functions than linear models can.