CNN Transfer Learning for Visual Guitar Chord Classification

Leon Tran
leontk@stanford.edu

Shawn Zhang
szhang22@stanford.edu

Eric Zhou
ericzhou@stanford.edu

1 Introduction

Automatic music chord recognition has always been a challenge. Recent attempts at recognizing notes on certain instruments have primarily utilized audio data. Despite early relative success in analyzing audio data via pitch profiles [3] and Hidden Markov Models [2], much of the progress around a purely aural approach has stagnated.

However, there is promise in visual recognition. Instead of focusing solely on audio, we wish to utilize music's natural visual aspect, starting with visually classifying guitar chords. We choose this because guitar chords are linked to unique hand shapes, which should allow a model to easily discriminate between multiple classes. Since deep visual chord recognition is a relatively unexplored field, applying CNNs to this task appears to be an impactful, novel application.

This project seeks to build a classifier to determine whether a guitarist is playing a C, D, Em (E minor), F, or G chord. The input to our system is a 1080×720 (grayscaled or RGB) image of a person playing a guitar. We then bound and crop the hands, passing the cropped image into our model to get an output prediction $\hat{y}$, where $\hat{y} \in \{C, D, Em, F, G\}$.

2 Related Work

2.1 Chord Recognition from Audio Data

Aural recognition of chords has been attempted numerous times. For example, Sheh et al. (2013) demonstrates chord prediction using a Hidden Markov Model, but found a lack of generalizability and poor performance of 40% accuracy [1]. The most successful approach is detailed in Fujishima et al. (1999), where a unique pitch profile was created for each chord and classification was attempted via a nearest-neighbors approach [2]. This method achieved an accuracy of 94%. While their model performed well, classification involved hand-crafted pitch profiles which is highly timbre-specific; audio-based approaches share this same pitfall. Because of this, visual chord recognition appears promising.

2.2 Chord Recognition from Visual Data

A number of groups have also tried visually analyzing guitar-playing albeit not for classifying chords. Kerdvibulvech et al. (2007) tracked the fingertips of the player to the guitar's frets; the team made predictions based off of these fingertip positions with 60.9% accuracy [3]. Meanwhile, Wang et al. (2018) utilized CNNs to segment hand images and used pose estimation to predict the quality of a guitarist's performance [7].

Although both papers use vision for guitar-playing, they do not directly look into visually distinguishing chords. Our project is the first to do so, where we use a CNN to capture the entire hand shape to predict the chord accordingly.

3 Datasets

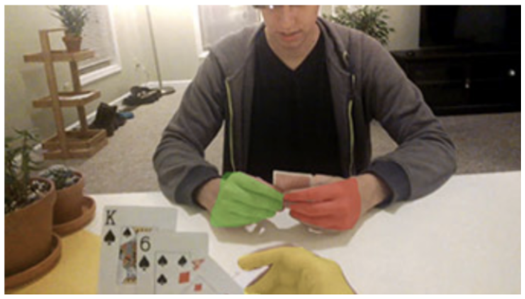


Figure 1: Example of EgoHands dataset.

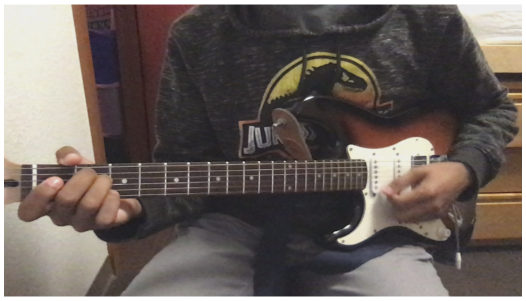


Figure 2: Example of original dataset (prior crop).

We used MobileNet v2 [4] trained on the EgoHands dataset [8] to extract hands from images [9]. The EgoHands dataset has 15,053 RGB videos with ground-truth, pixel-level hand labelings.

For our classification task, we built our own original dataset by videotaping five different people each playing five basic chords: C, D, Em, F, and G. With over 2 hours of footage collected, we then spliced timeframes into individual 1080×720 RGB still images. Later, we ran them through MobileNet v2 to bound and crop the hands from the images. In total, we had 331 C chord images, 324 G chord images, 393 D chord images, 298 Em chord images, and 450 F chord images. We randomly split these 1797 data into 1437 (80%) examples, 180 (10%) examples, and 180 (10%) examples for the training, validation, and test sets, respectively.

3.1 Data Augmentation

First, we resized the hand images from our original dataset to 224×224 and normalized each color channel by values from ImageNet (to fit into our pre-trained models in Section 4). To counteract the limitedness and similarity of our data, we applied random horizontal flipping and random rotating with angle θ, where $\theta \in [-30^\circ, 30^\circ]$. A grayscale copy of the dataset was also generated for experimentation in Section 5.

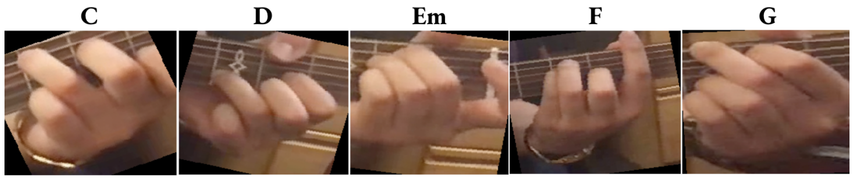


Figure 3: Chords post-augmentation. Note: colors aren't normalized here for the clarity of the reader.

4 Methods and Models

4.1 Methodology

Due to our limited data, an end-to-end pipeline was not feasible. Instead, we broke it down into two: (1) One pipeline took full images into MobileNet v2 to extract only the fretting hand (see Section 3). (2) We then fed those images into our second pipeline, the "main" neural network for classification.

Again from our limited data, we were inspired to use transfer learning for our main CNN by taking weights from either ResNet18 or GoogLeNet (where both were trained on ImageNet).

For each network in training, we flattened the output of the convolutional layer to generate an n_F dimensional layer. We then appended 3 fully-connected layers of size $(n_F \times 64)$, (64×32), and (32×5) respectively for our specific task. Additionally, we unfroze up to two of the last layers in the original network prior training. Finally, both networks aimed to minimize cross-entropy loss with the ADAM optimization algorithm.

4.2 Models

4.2.1 ResNet18

ResNet18 is a state-of-the-art convolutional network architecture with 18 convolutional layers of varying sizes [5]. The ResNet18 architecture also features skip connections, which adds the output of the l^{th} layer $a^{[l]}$, to the linear output of the $l + 2^{th}$ layer, $z^{[l+2]}$, for certain l. This connection allows the network to learn the identity function more simply, improving network performance. This is because we may choose to "exclude" any hidden layers by learning the identity function.

4.2.2 GoogLeNet

GoogLeNet is another state-of-the-art convolutional neural network architecture, containing 22 trainable layers and 5 max-pooling layers [6]. The network contains Inception modules which make up the majority of the layers. Inception modules essentially allow multiple convolutions of the same input image with different sized kernels. The output volumes of those convolutions are concatentated and fed into the next layer. In this network, we convolve the image with 5×5, 1×1, max-pooling followed by 1×1, and 3×3 filters with stride 2.

5 Experimentation

In our experimentation, we searched for which hyperparameters yielded the best accuracy. Specifically, we checked for RGB/grayscale, learning rate, number of unfrozen layers, and weight decay.

As such, we trained our ResNet18 classifier on both an RGB and grayscaled version of the dataset. After noticing that RGB produced considerably greater results than Grayscale, the GoogLeNet classifier was trained only on the RGB version of our dataset. Two different learning rates, 1E-03 and 1E-04 were tested for all cases. In addition to this, for each model, we tried unfreezing 0, 1, or 2 of the last layers. Finally, each model was trained and tested both without weight decay and with a weight decay of λ =1E-05.

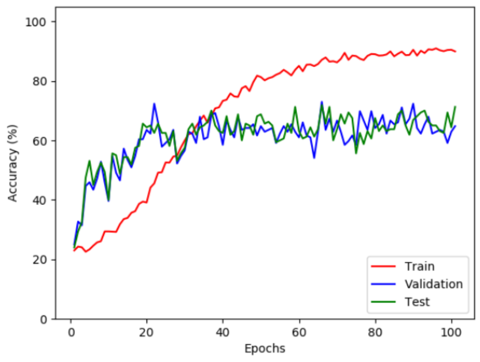


Figure 4: ResNet18 RGB Training.

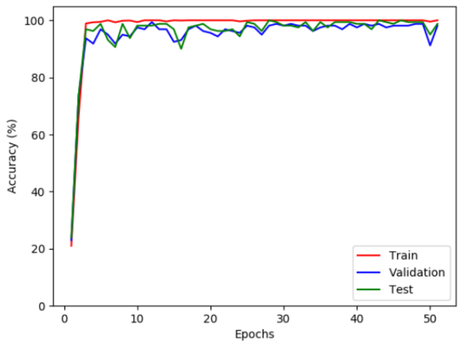


Figure 5: GoogLeNet RGB Training.

Note: ResNet18 has initially worse train accuracy than val/test, which is due to dropout.

6 Results and Discussion

Here, we display the results of our experimentation:

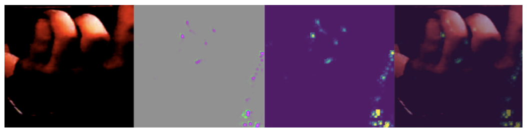


Figure 6: ResNet18 RGB Saliency Map.

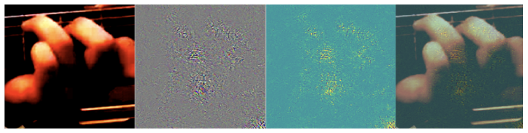
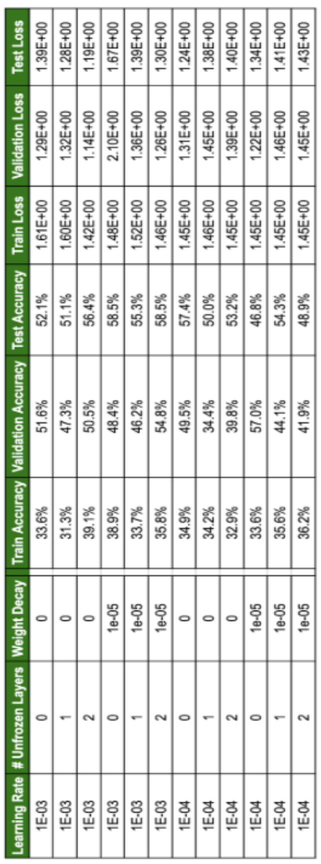
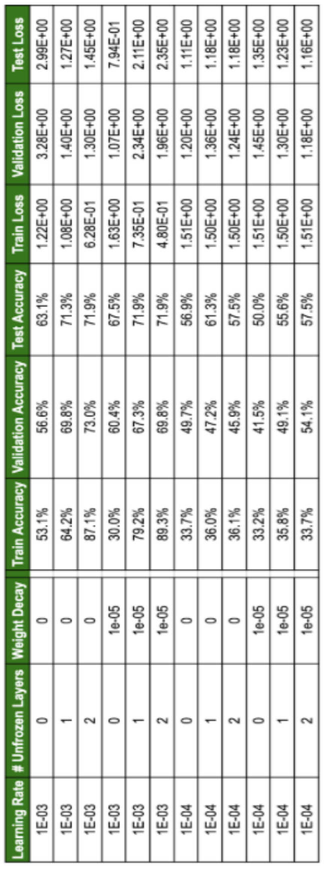


Figure 7: GoogLeNet RGB Saliency Map.



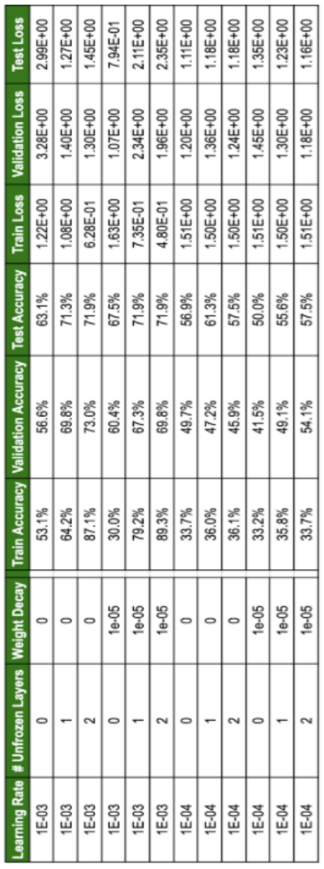
Learning Rate	# Unfrozen Layers	Weight Decay	Train Accuracy	Validation Accuracy	Test Accuracy	Train Loss	Validation Loss	Test Loss
1E-03	0	0	33.6%	51.6%	52.1%	1.61E+00	1.29E+00	1.39E+00
1E-03	1	0	31.3%	47.3%	51.1%	1.60E+00	1.32E+00	1.28E+00
1E-03	2	0	39.1%	50.5%	56.4%	1.42E+00	1.14E+00	1.19E+00
1E-03	0	1e-05	38.9%	48.4%	58.5%	1.48E+00	2.10E+00	1.67E+00
1E-03	1	1e-05	33.7%	46.2%	55.3%	1.52E+00	1.36E+00	1.39E+00
1E-03	2	1e-05	35.8%	54.8%	58.5%	1.46E+00	1.26E+00	1.30E+00
1E-04	0	0	34.9%	49.5%	57.4%	1.45E+00	1.31E+00	1.24E+00
1E-04	1	0	34.2%	34.4%	50.0%	1.46E+00	1.45E+00	1.38E+00
1E-04	2	0	32.9%	39.8%	53.2%	1.45E+00	1.39E+00	1.40E+00
1E-04	0	1e-05	33.6%	57.0%	46.8%	1.45E+00	1.22E+00	1.34E+00
1E-04	1	1e-05	35.6%	44.1%	54.3%	1.45E+00	1.46E+00	1.41E+00
1E-04	2	1e-05	36.2%	41.9%	48.9%	1.45E+00	1.45E+00	1.43E+00

Figure 8: ResNet18 Grayscale.



Learning Rate	# Unfrozen Layers	Weight Decay	Train Accuracy	Validation Accuracy	Test Accuracy	Train Loss	Validation Loss	Test Loss
1E-03	0	0	53.1%	56.6%	63.1%	1.22E+00	3.28E+00	2.99E+00
1E-03	1	0	64.2%	69.8%	71.3%	1.08E+00	1.40E+00	1.27E+00
1E-03	2	0	87.1%	73.0%	71.9%	6.28E-01	1.30E+00	1.45E+00
1E-03	0	1e-05	30.0%	60.4%	67.5%	1.63E+00	1.07E+00	7.94E-01
1E-03	1	1e-05	79.2%	67.3%	71.9%	7.35E-01	2.34E+00	2.11E+00
1E-03	2	1e-05	89.3%	69.8%	71.9%	4.80E-01	1.96E+00	2.35E+00
1E-04	0	0	33.7%	49.7%	56.9%	1.51E+00	1.20E+00	1.11E+00
1E-04	1	0	36.0%	47.2%	61.3%	1.50E+00	1.36E+00	1.18E+00
1E-04	2	0	36.1%	45.9%	57.5%	1.50E+00	1.24E+00	1.18E+00
1E-04	0	1e-05	33.2%	41.5%	50.0%	1.51E+00	1.45E+00	1.35E+00
1E-04	1	1e-05	35.8%	49.1%	55.6%	1.50E+00	1.30E+00	1.23E+00
1E-04	2	1e-05	33.7%	54.1%	57.5%	1.51E+00	1.18E+00	1.16E+00

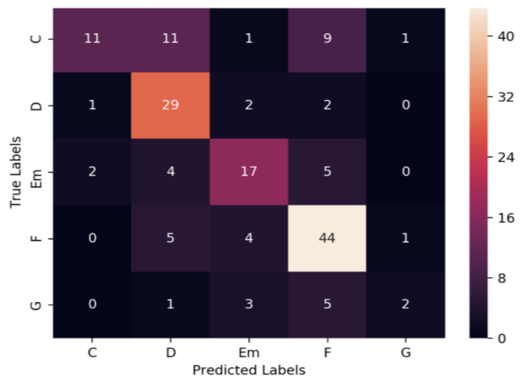
Figure 9: ResNet18 RGB.



Learning Rate	# Unfrozen Layers	Weight Decay	Train Accuracy	Validation Accuracy	Test Accuracy	Train Loss	Validation Loss	Test Loss
1E-03	0	0	53.1%	56.6%	63.1%	1.22E+00	3.28E+00	2.99E+00
1E-03	1	0	64.2%	69.8%	71.3%	1.08E+00	1.40E+00	1.27E+00
1E-03	2	0	87.1%	73.0%	71.9%	6.28E-01	1.30E+00	1.45E+00
1E-03	0	1e-05	30.0%	60.4%	67.5%	1.63E+00	1.07E+00	7.94E-01
1E-03	1	1e-05	79.2%	67.3%	71.9%	7.35E-01	2.34E+00	2.11E+00
1E-03	2	1e-05	89.3%	69.8%	71.9%	4.80E-01	1.96E+00	2.35E+00
1E-04	0	0	33.7%	49.7%	56.9%	1.51E+00	1.20E+00	1.11E+00
1E-04	1	0	36.0%	47.2%	61.3%	1.50E+00	1.36E+00	1.18E+00
1E-04	2	0	36.1%	45.9%	57.5%	1.50E+00	1.24E+00	1.18E+00
1E-04	0	1e-05	33.2%	41.5%	50.0%	1.51E+00	1.45E+00	1.35E+00
1E-04	1	1e-05	35.8%	49.1%	55.6%	1.50E+00	1.30E+00	1.23E+00
1E-04	2	1e-05	33.7%	54.1%	57.5%	1.51E+00	1.18E+00	1.16E+00

Figure 10: GoogLeNet RGB.

Note: Each table row displays the results where the maximum test accuracy was achieved.



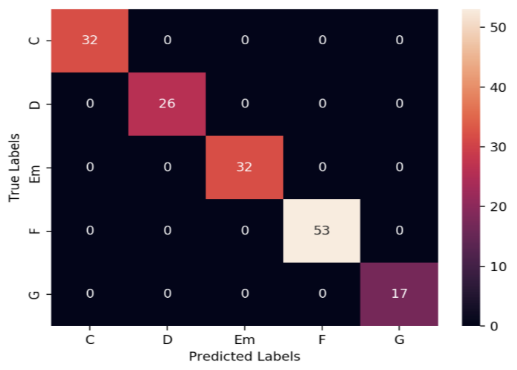


Figure 11: ResNet18 RGB Confusion Matrix. Figure 12: GoogLeNet RGB Confusion Matrix.

Note: The confusion matrices display the results from the test set.

6.1 Model Comparison

GoogLeNet attained a maximum accuracy of 100% while ResNet 18 achieved a maximum accuracy of 71% on our test set. Since our data are highly correlated and look perhaps too similar to each other, we believe that the training set matches the test set too well. Although GoogLeNet performed extremely well, the data does not represent real-world situations, making our test set error artificially low. The fact that GoogLeNet outperformed ResNet18 may be due to the flexibility of the inception architecture and GoogLeNet's lower number of parameters leading to different features downstream.

The models trained on the RGB dataset performed better than those trained on the Grayscale dataset. We believe this to be due to the loss of information through contrast in the grayscaling process. Additionally, the pre-trained CNNs were trained originally on RGB images, so there may be loss in transferability.

6.2 Model Robustness

GoogLeNet yielded high training and test accuracy on our training and test sets. On new "realistic" examples not taken from our dataset, the model was only accurate for F and Em chords. The poor generalization is a result of our dataset not being diverse. The training examples are from similar angles, resolutions, and background lighting conditions. This could be fixed by having a more diverse training set or additional methods of image augmentation. Secondarily, we could have applied regularization methods that added more noise to help support generalization.

6.3 Model Analysis

In terms of hyperparameter tuning, a smaller learning rate yielded lower test accuracy for ResNet18 RGB, whereas it had the opposite effect on GoogLeNet. There seemed to be no impact on ResNet18 trained on grayscaled data.

As the number of unfrozen layers increased, ResNet18 was more prone to overfit to the train data. This is also because ResNet only contains 18 layers, compared to GoogLeNet's 27 layers; this means that unfreezing the last few layers of ResNet 18 would have a greater impact on its final accuracy. However, there was no clear impact on GoogLeNet's performance, probably due to its already high accuracy with zero unfrozen layers and its larger number of layers.

The presence of weight decay on both models had an indeterminate effect. Ultimately, the architecture search had the largest impact on test accuracy. That said, within the various experiments run on GoogLeNet, the version trained with 0 additional unfrozen layers, no weight decay, and a learning rate of 1E-04 achieved the highest accuracy and lowest test loss.

The ResNet confusion matrix indicates that the C chord is most often misclassified with D and F, which may be because the shape is less unique and resembles those other chords. The ResNet model is also poor at predicting G chords, which may come from the lack of G-chord data.

The saliency maps are particularly interesting. The ResNet18 saliency map reveals that it puts a lot of weight into a select few pixels, which seem to outline the hand's edges. Meanwhile, the GoogLeNet saliency map appears to focus the shape of the hand and joints/knuckles. The latter seems more intuitive for identifying the uniqueness of a chord.

7 Future Work

While a high accuracy on the train and test set was achieved through transfer learning on GoogLeNet, there remains much progress to be made. Future areas of research could include improving model robustness through the collection of a more diverse dataset composed of examples with more diversity in hand color, hand size, image angle, and background. It may also be worth doing a more thorough search of CNN architectures with the new dataset. Accuracy could potentially be improved further through an ensemble approach, by incorporating audio data into the classification pipeline. Finally, further optimization could be looked into so that the system could work in real-time on video data, creating a more realistic testing environment.

8 Contributions

Leon Tran set up MobileNetv2 to extract hand images and generate and augment the data for the project. Shawn Zhang performed transfer learning on GoogLeNet/ResNet18 for classification. Eric Zhou helped with the poster and report.

References

[1] Sheh, A. and Ellis, D.P.W. “Chord Segmentation and Recognition Using EM-Trained Hidden Markov Models”, *Proc. ISMIR*, 2003.

[2] Fujishima, T. “Realtime Chord Recognition of Musical Sound: a System Using Common Lisp Music”, *Proc. ICMC*, pp.464–467, 1999.

[3] Kerdvibulvech, Chutisant, and Hideo Saito. "Vision-based detection of guitar players’ fingertips without markers." *Computer Graphics, Imaging and Visualisation (CGIV 2007).* IEEE, 2007.

[4] Sandler, Mark, et al. "Mobilenetv2: Inverted residuals and linear bottlenecks." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2018.

[5] He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2016.

[6] Szegedy, Christian, et al. "Going deeper with convolutions." *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2015.

[7] Wang, Z, et al. "A 3D Guitar Fingering Assessing System Based on CNN-Hand Pose Estimation and SVR-Assessment." *Society for Imaging Science and Technology.* 2018.

[8] "EgoHands: A Dataset for Hands in Complex Egocentric Interactions." *Indiana University Computer Vision Lab*. 2017.

[9] Dibia, V. "HandTrack: A Library For Prototyping Real-time Hand Tracking Interfaces using Convolutional Neural Networks". *Github*. 2017