Musical Instrument Classification Using a Hybrid Neural Network

Landon Buell

B.S. Physics, Dec 2020

M.S. Computer Science, Exp. 2024

University of New Hampshire

Kevin Short

University Professor & Professor of Mathematics

Founder, Integrated Applied Mathematics Program

University of New Hampshire

8 Jan 2021

Presentation Outline

Introduce the *problem* that we are going to solve

Develop *Neural Networks* as the solution

Discuss consequences and *improvements* to the solution

Analyze the *performance* of the improvements

Introduction

Mapping Sounds to Sources

Humans are proficient at mapping sounds to sources

Impractical at a large scale

Computers are not proficient at mapping sounds to sources

Can handle large volumes of data

“Birds inspired us to fly, burdock plants inspired Velcro and nature has inspired many other inventions. It seems only logical then, to look to the brain’s architecture for inspiration on how to build an intelligent machine.”

- Aurelion Geron, Former YouTube Video Classification lead

The Neural Network

Structure

Consider a neural network to be just like a mathematical function

Composed of smaller functions called *layers*

Transform *features* into *predictions*

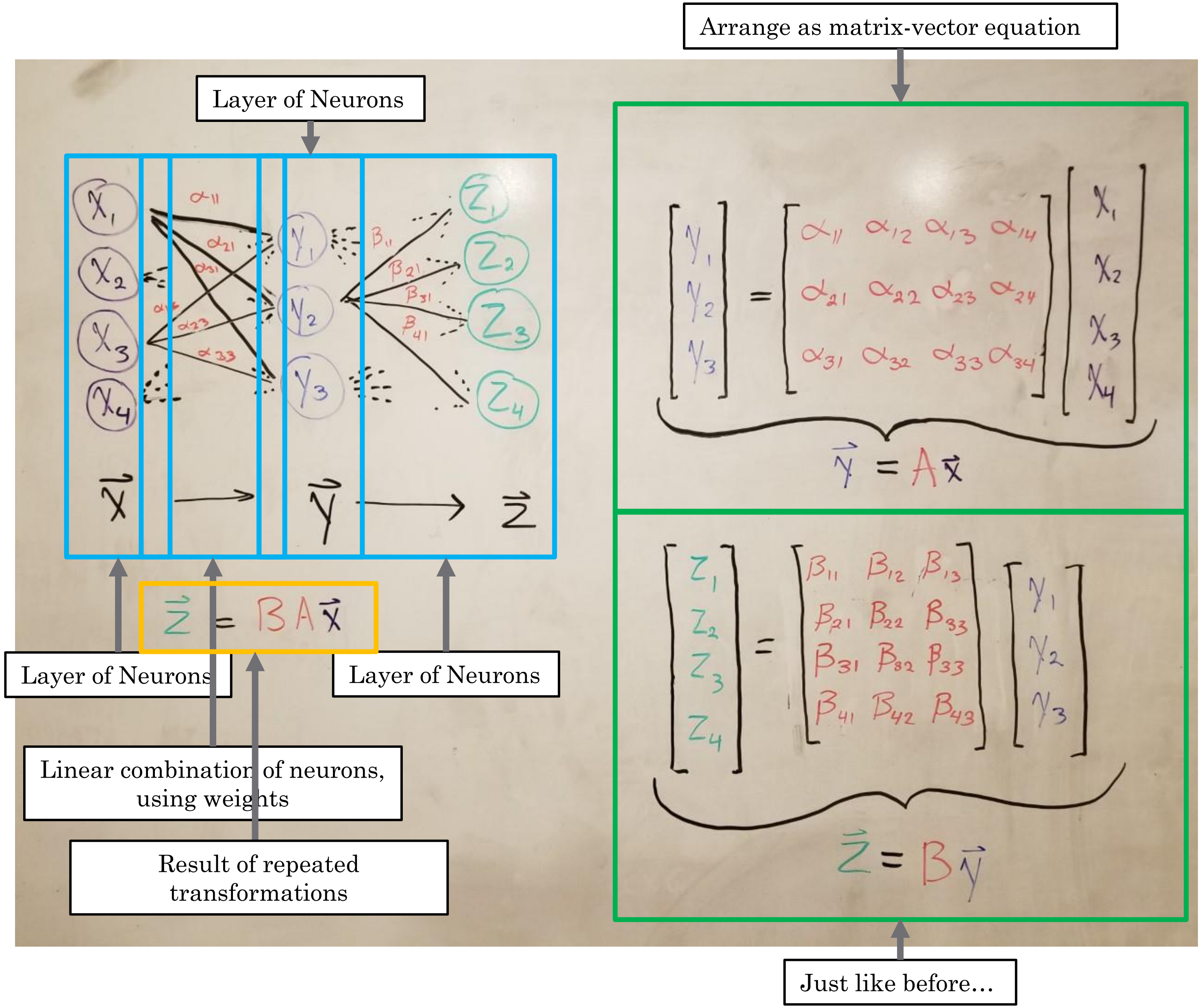
Classification Model

Inputs are properties of digital audio files from London's *Philharmonia* Orchestra and University of Iowa's *Electronic Music Studios*

Outputs are integers that correspond to musical instruments

We group samples with similar input properties

The Multilayer Perceptron (MLP)



- Connect layers with *weights*
- *Dense* Connectivity
- Handles 1D samples

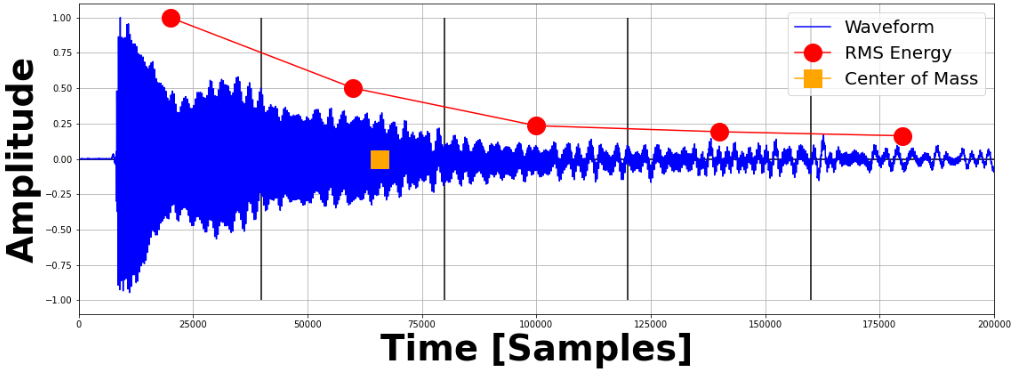
MLP (Cont.)

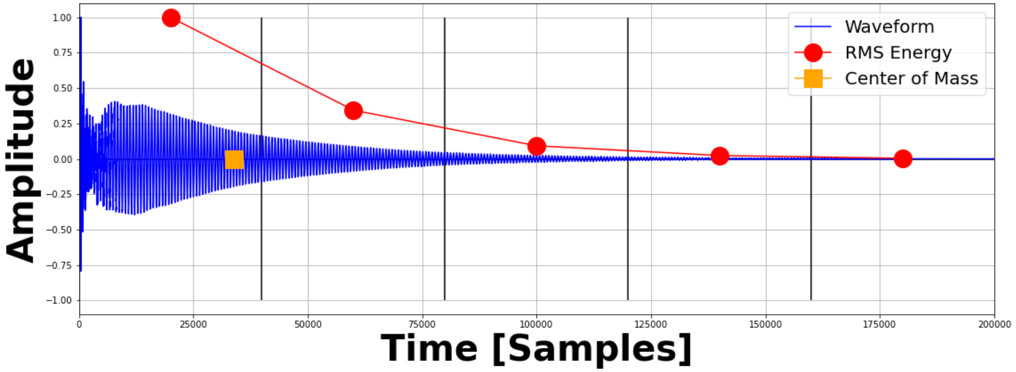
- Formal implementations add a *bias* and an *activation function*

$$\vec{x}^{(l)} = \sigma^{(l)}\left(\underbrace{W^{(l)}\vec{x}^{(l-1)}}_{\text{Matrix-vector product}} + \vec{b}^{(l)}\right)$$

- Allows us to model more complex decision boundaries in real-world problems

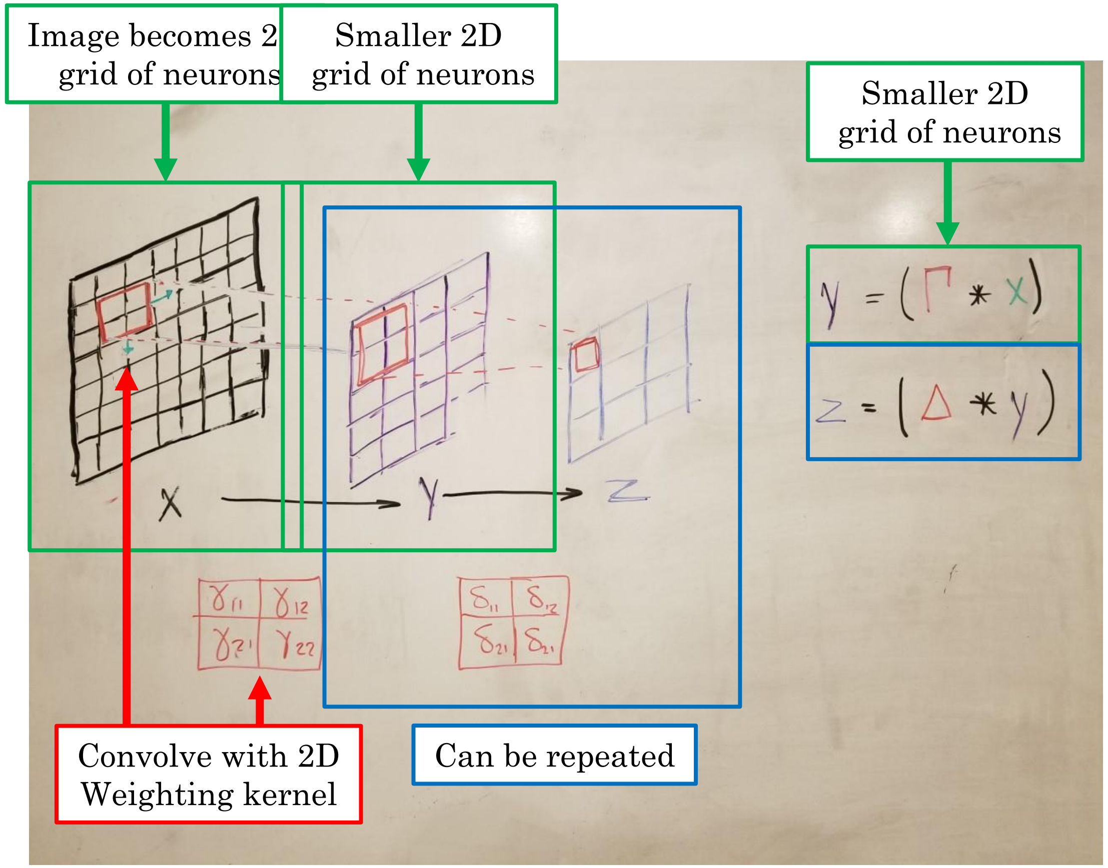
Features for the MLP





- Time Domain Envelope (x5)
- Zero Crossing Rate
- Temporal Center of Mass
- Auto Correlation Coefficients (x4)
- Mel Frequency Cepstrum Coefficients (x12)
- Frequency Center of Mass

The Convolutional Neural Network (CNN)



- Connect layers with weighting kernels
- *Sparse* connectivity
- Handles 2D "grid" of neurons (images)
- Followed by layer of pooling

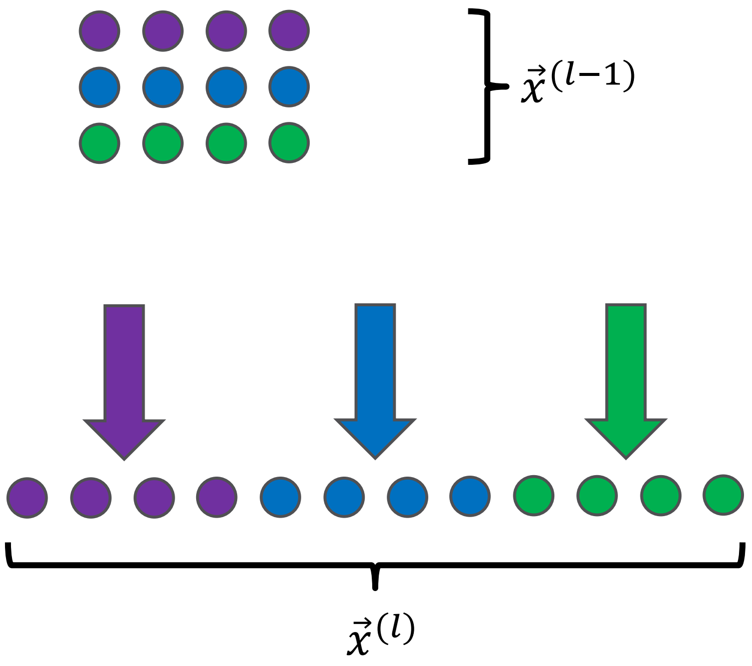
CNN (Cont.)

- Formal implementation also add a *bias* and an *activation function*

$$x^{(l)}[i,j] = \sigma^{(l)}\left(b^{(l)} + \underbrace{\sum_u \sum_v W^{(l)}[i,j]x^{(l-1)}[i-u,j-v]}_{\text{2D Convolution operation}}\right)$$

- Detect patterns and features in images, using compressed versions of a parent image

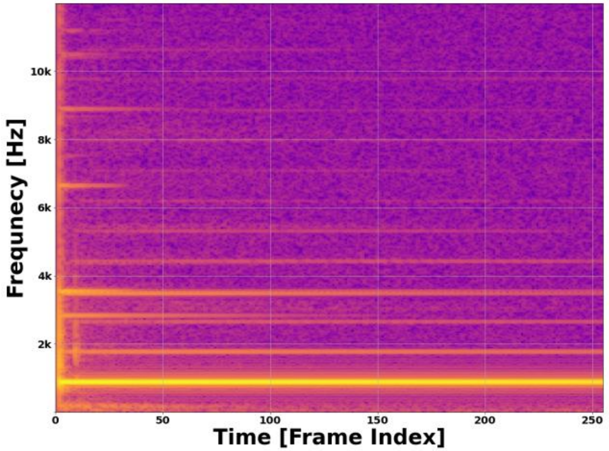
The Flattening Operation



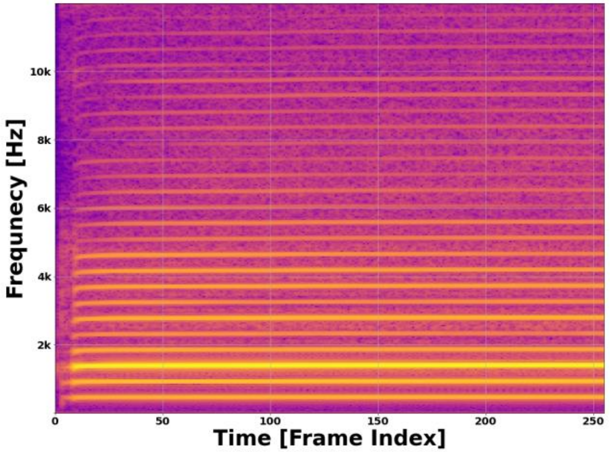
- Transforms 2D grid of neurons to a 1D row/col of neurons
- Allows images to be passed into dense layers

Features for the CNN

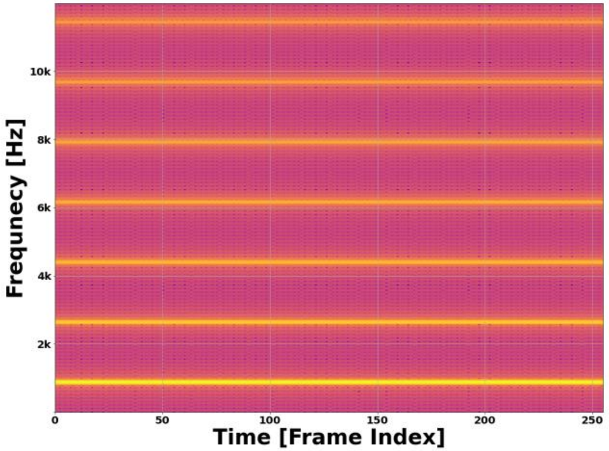
Vibraphone - A5



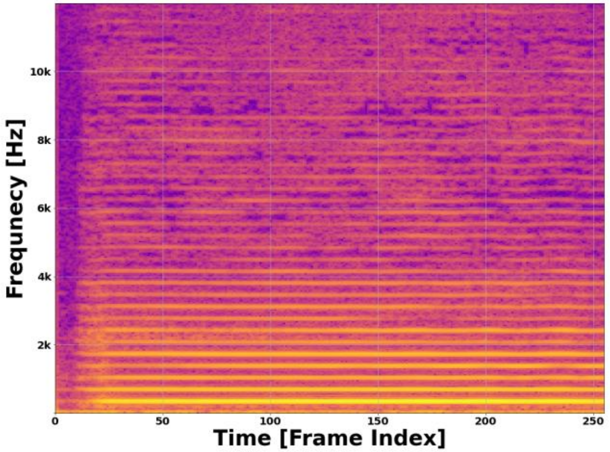
Oboe – Bb4



Square Wave - A5



Cello – F4

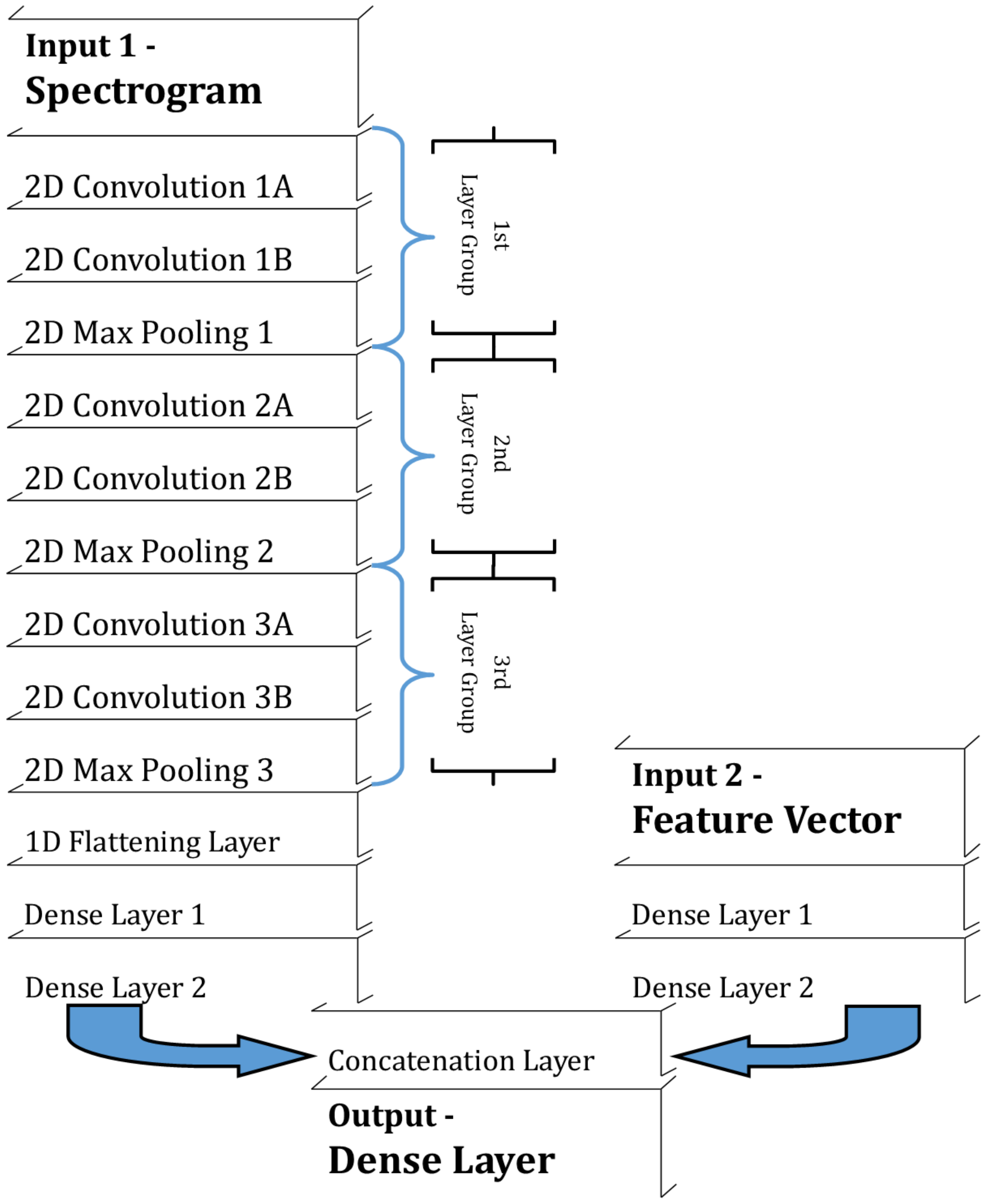


Consequence of the Solutions

- Both architectures have experimentally shown individual success
- MLP and CNN use different forms of inputs that are initially *incompatible*
  - 1D vs. 2D Inputs
- We can *combine* them to form a *Hybrid* Neural Network
  - Connections made at hidden layer

Hybrid Network Architecture

Hybrid Neural Network Architecture



Implementation (Tensorflow.keras)

```python
@staticmethod
def MultiInputNetwork (name,inputA,inputB,n_classes):
    """
    Create Multi-Input layer neural Network
    --------------------------------
    name (str) : Name to use for neural network
    inputA (iter) : list-like of ints indicating input shape of CNN branch
    inputA (iter) : list-like of ints indicating input shape of MLP branch
    n_classses (int) : Number of unique output classes
    --------------------------------
    Return complied tf.keras model
    """
    modelCNN = NeuralNetworkModels.ConvolutionalNeuralNetwork2D(inputA,n_classes,
                    filterSizes=[32,32,32],kernelSizes=[(3,3),(3,3),(3,3)],
                    poolSizes=[(3,3),(3,3),(3,3)],neurons=[64,64])
    modelMLP = NeuralNetworkModels.MultilayerPerceptron(inputB,n_classes)

    x = keras.layers.concatenate([modelCNN.output,modelMLP.output])

    x = keras.layers.Dense(units=n_classes,activation='softmax',name='Output')(x)
    modelMain = keras.Model(name=name,inputs=[modelCNN.input,modelMLP.input],outputs=x)

    modelMain.compile(optimizer=keras.optimizers.Adam(learning_rate=0.001,
                                beta_1=0.9,beta_2=0.999,epsilon=1e-07),
                    loss=keras.losses.CategoricalCrossentropy(),
                    metrics=[keras.metrics.Accuracy(),keras.metrics.Precision(),keras.metrics.Recall()])
    return modelMain
```

Create each input Branch as its own model

Create combined output layer, wrap as single model instance

Return the hybrid network instance

Concatenation of each model's output activations

Compile the model, prepare for training/testing

This is called *Multimodal* or M*ultiview* Deep Learning

Transform incompatible inputs to a compatible format at an internal layer

Performance of the Hybrid Model

Evaluating a Model

WE *TRAIN* A MODEL ON A *SUBSET* OF ALL SAMPLES

THE MODEL *LEARNS* A SET OF PARAMETERS IN EACH LAYER THAT ALLOWS THE MAPPING OF *FEATURES* TO *PREDICTIONS*

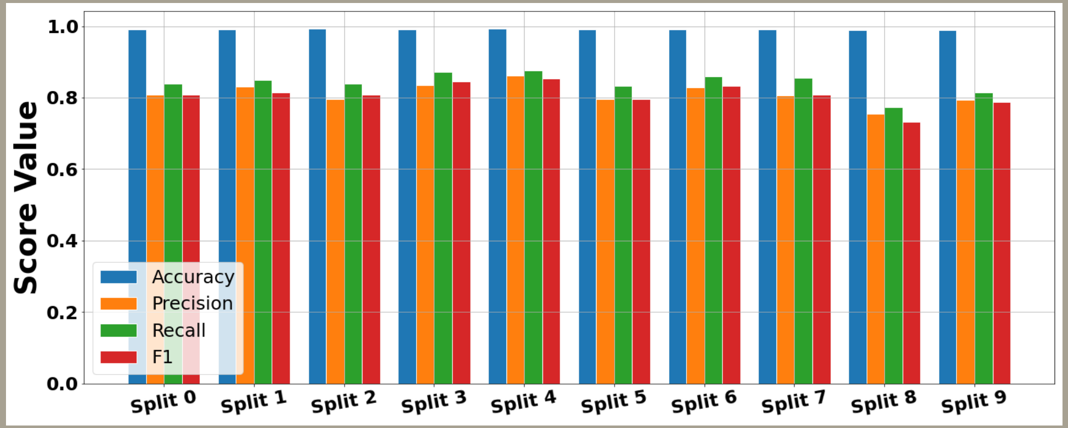
WE *TEST* THE MODEL ON THE REMAINING SAMPLES THAT THE MODEL HAS *NEVER INTERACTED* WITH

X-Validation Performance

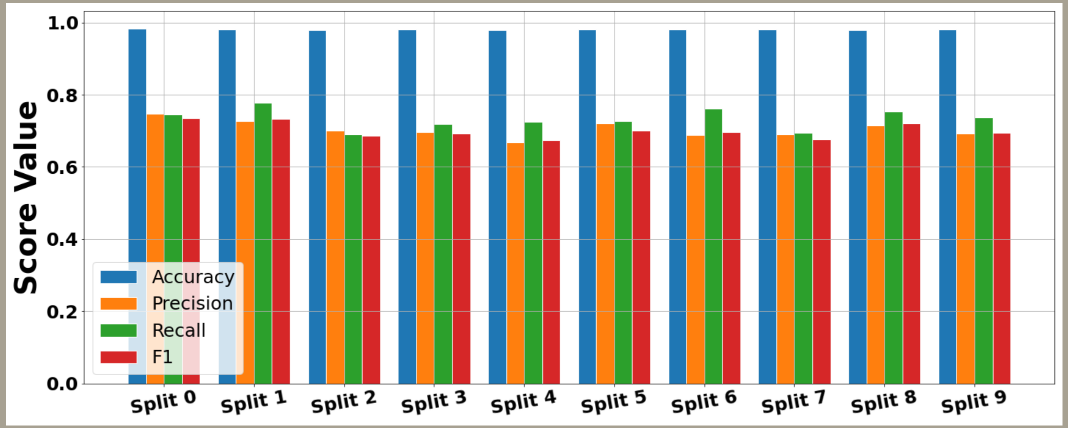
- Compare three variants
  1) CNN Architecture
  2) MLP Architecture
  3) Hybrid Architecture

- Hybrid network shows generally better and more consistent performance scores

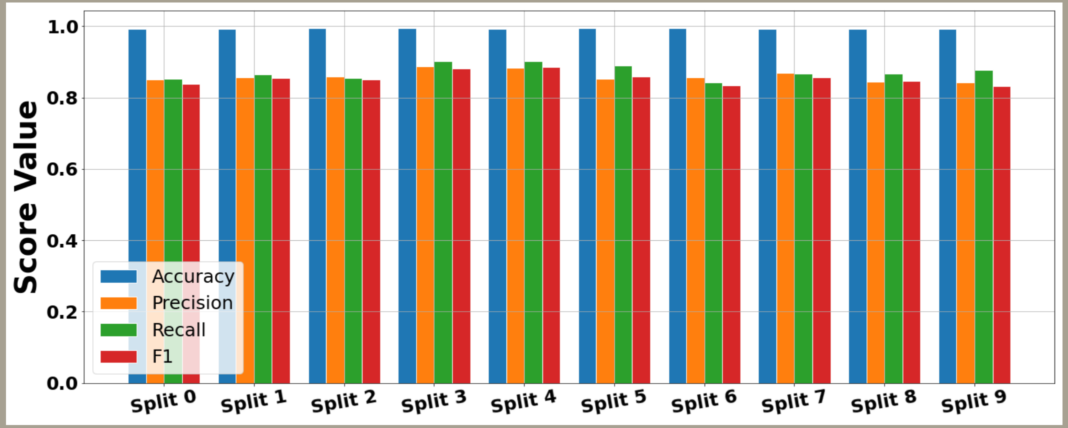
1)



2)

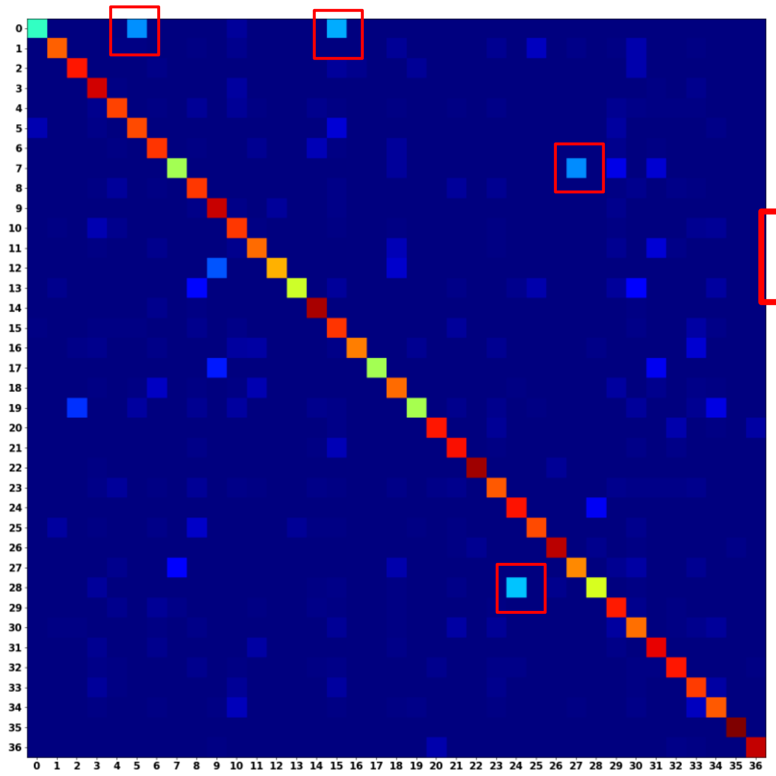


3)



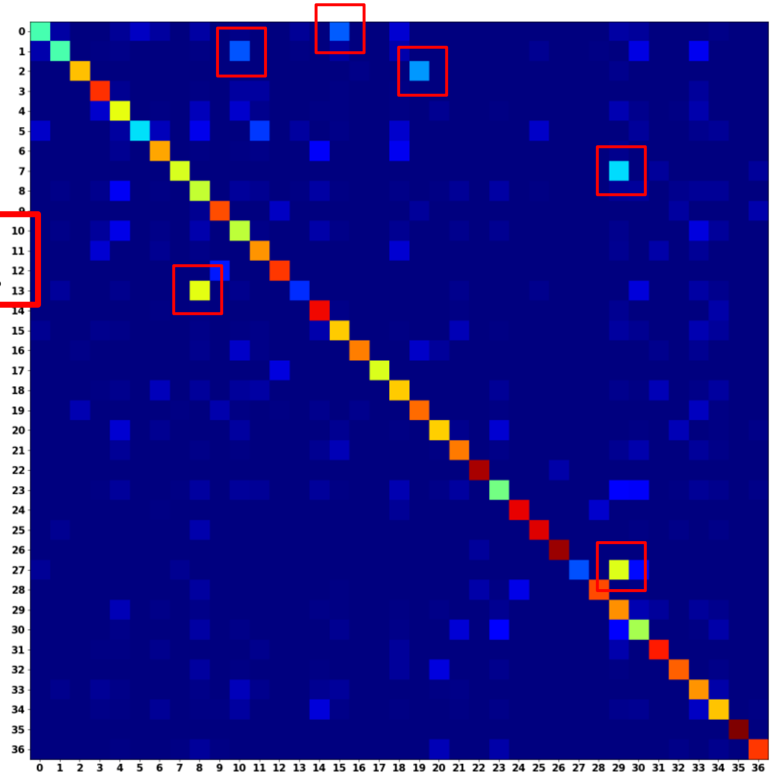
Single NN Confusion Matrices

CNN Branch (Occurrence Weighted)



Strong main diagonal, reasonable performance

MLP Branch (Occurrence Weighted)

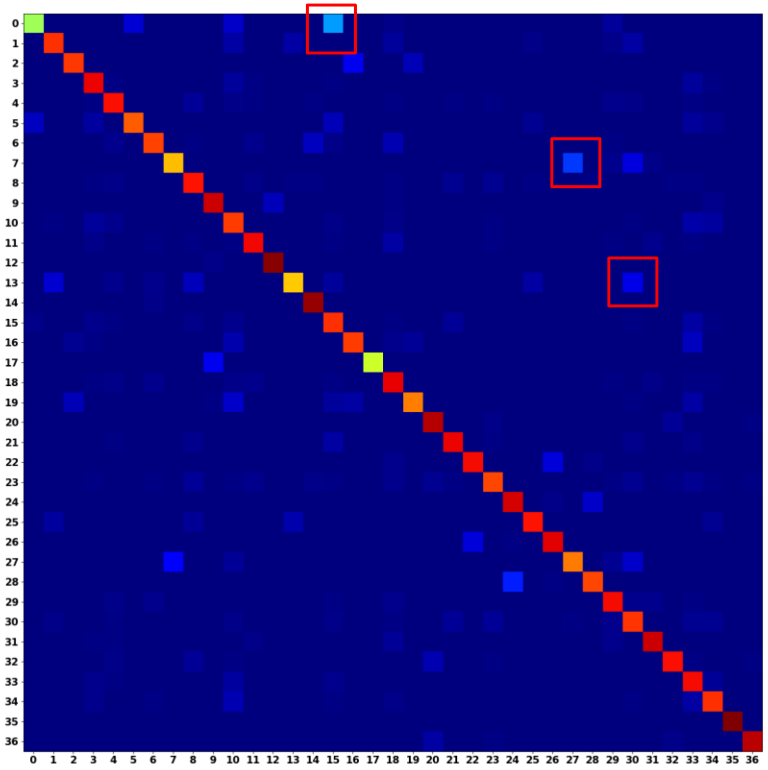


Frequent misclassifications

Related classes?
Poor features choice?

Hybrid NN Confusion Matrix

CNN + MLP (Occurrence Weighted)



- Hybrid model shows a stronger main diagonal
- Combines predictive power from both architectures

Less Frequent, Less
Pronounced misclassifications

Discussion

- Hybrid networks allows for *multimodal learning*
  - Transform inputs into compatible formats
  - Combine multiple inputs to form one prediction
- Opens the door to other related classification tasks
- Many uses outside digital audio recognition
  - Cardiograph + Biometric readings
  - Speech + text information
  - Video + Audio

- The *hybrid network* shows improved classification performance over either unimodal model

- Possibility to generalize to other related tasks

Conclusions

Refrences

[1] Geron, Aurelien. Hands-on Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly, 2017.

[2] Goodfellow, Ian, et al.Deep Learning. MIT Press, 2017.

[3] Khan, M. Kashif Saeed, and Wasfi G. Al-Khatib. "Machine-Learning Based Classification of Speech and Music." Multimedia Systems, vol. 12, no. 1, 2006, pp. 55–67., doi:10.1007/s00530-006-0034-0.

[4] Li, Yingming, and Ming Yang. "A Survey of Multi-View Representation Learning." Journal of LateX Class Files, vol. 14, no. 8, Aug. 2015.

[5] Liu, Zhu, et al. "Audio Feature Extraction and Analysis for Scene Segmentation and Classification." Journal of VLSI Signal Processing, vol. 20, 1998, pp. 61–79.

[6] Ngiam, Jiquan, et al. "Multimodal Deep Learning." 2011.

[7] TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[8] Virtanen, Tuomas, et al. Computational Analysis of Sound Scenes and Events. Springer, 2018.

[9] White, Harvey Elliott, and Donald H. White. Physics and Music: the Science of Musical Sound. Dover Publications, Inc., 2019.

Additional information, GitHub Repository, and Formal Write-Up is Available upon request

- Landon Buell – lhb1007@wildcats.unh.edu
- Kevin Short – kevin.short@unh.edu

Thank you very much!

Questions?

Appendix – Confusion Matrices

Standard Confusion Matrix

- For k categories, is a k x k matrix

$C[i, j]$ = Number of samples that belong to class i but were predicted to be in class j

- A strong classifier has a dominant main-diagonal

Occurrence Weighted Matrix

- Created same as standard matrix
- Divide each row by sum of the row
- Accounts for non-uniform number of samples in each class

Appendix - Metrics

Precision / Specificity Score

- Bound on [0,1] – higher is favorable

$$Precision = \frac{TP}{TP + FP}$$

- *"How many selected items are relevant to the problem?"*

Recall / Sensitivity Score

- Bound on [0,1] – higher is favorable

$$Recall = \frac{TP}{TP + FN}$$

- *"How many relevant items to the problem have been selected?"*

Appendix – Metrics (Cont.)

Accuracy Score

- Ratio of correct predictions to total predictions

$$Accuracy = \frac{TP + FN}{TP + FP + FN + TP}$$

- Does not account for non-uniform number of samples in each class

F1 Score

- Harmonic Mean of Precision and Recall scores

$$F1 = 2 \times \frac{Precision + Recall}{Precision \times Recall}$$

- Favors models with high precision *and* high recall

Appendix – Activation Functions

- Activation functions are typically included as a “last” step in each layer function
- Can take many forms based on network or layer type
  - ReLU for classification
  - Sigmoid for regression
  - Softmax for normalized output
- Turns affine-transform into non-linear transform
- Model more complex solution spaces

Appendix – Tensorflow / Keras

- Free Python library centered around Deep Learning
- High-Level API for constructing neural networks and computational graphs
- Developed and maintained by Google
  https://www.tensorflow.org/
- Offers tools for hardware acceleration on certain GPUs

Appendix – Musical Instrument Samples

- University of Iowa Electronic Music Studios

  [University of Iowa Electronic Music Studios (uiowa.edu)]

- Philharmonia Symphony Orchestra

  https://philharmonia.co.uk/

- Samples preprocessed
  - *WAV* file format
  - 44.1 kHz sample rate
  - Mono-channeled
  - 16 bit-depth

- Features extracted with Python program available on GitHub