Get Dataset

- 32 subjects
- 40 sets of 63 second recordings (first 3 seconds are buffer)
- 512 Hz Recording
- 32 EEG channels
- (30×40) 1280 recordings 32×(512×63) 32256 matrices

Input EEG: 1280 × 32 × 32256 matrix (epochs, channels, data points in recording)

Output: 1280 × 2 matrix (epochs, values, arousal)

Remove first 3 seconds of buffer time from each epoch
(512×63) => (512×60)
32256 => 30720

1280 × 32 × 30720 matrix

Decimate Data
(512×60) => (128×60)
30720 => 7680

1280 × 32 × 7680 matrix

General Band-Pass Filter (0.1→50 Hz)
- this is used to get rid of general noise

1280 × 32 × 7680 matrix

Butterworth Band-Pass Filter
α (1-7 Hz), β (8-13 Hz), θ (14-30 Hz), γ (30-45 Hz)
- This will break up the single stream of data from each channel into 4 streams of separate bands

1280 × 4 × 32 × 7680 matrix (four different bands)

Epoch the Epochs
- For each trial of 60s, 14 segments are obtained using an 8s time window moving every 4s.

- This extrapolates our training data
- (128×60) => (128×8)
7680 => 1024
- 1280 × 14 = 17920

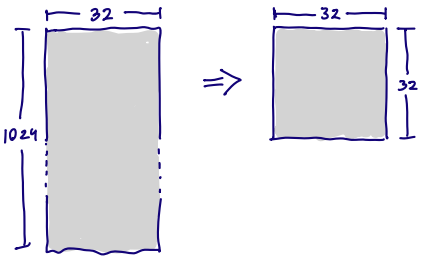
1280 × 2 matrix

- Increasing our input data, means we increase output
- All 14 new samples from an epoch have the same output

17920 × 2 matrix

17920 × 4 × 32 × 1024 matrix

PCC Feature Extraction
- Pearson Correlation Coefficient
- On every band and epoch
- Basically calculating correlation of every channel pair against each other



17920 × 4 × 32 × 32 matrix

80% Training Dataset

20% Test Dataset

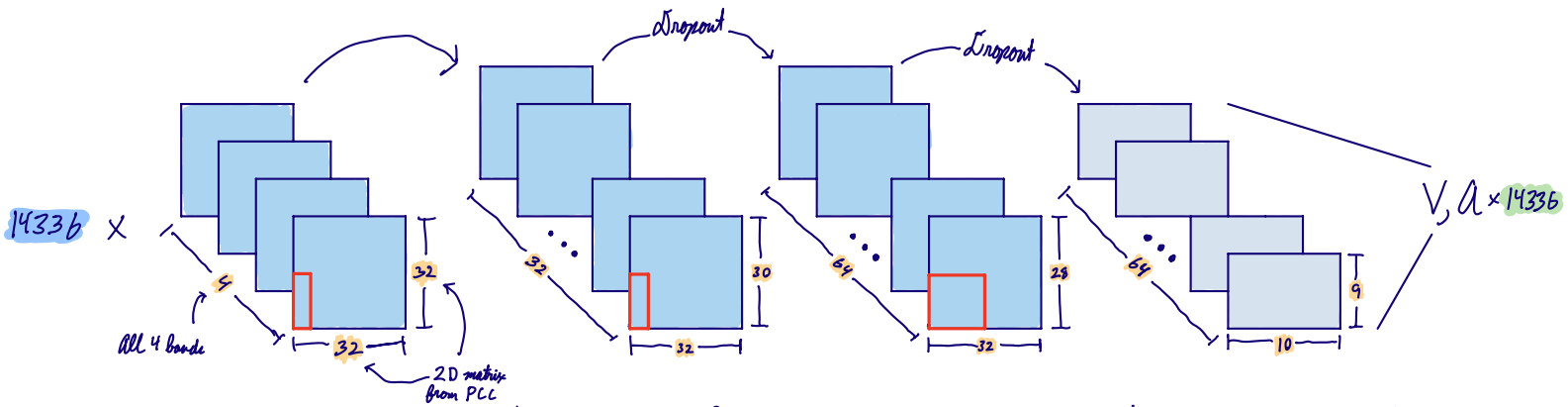
14336 × 4 × 32 × 32 Training Matrix

3584 × 4 × 32 × 32 Testing Matrix

14336 × 2 Output Training Matrix → 3584 × 2 Output Testing Matrix

Train Convolutional Neural Network (CNN)

- Training Epochs = 50
- Batch Size = 128
- Learning Rate = 0.01



Input Convolutional Layer
- Kernel size = 3×1
- # of Kernels = 32

Convolutional Layer
- Kernel size = 3×1
- # of Kernels = 32

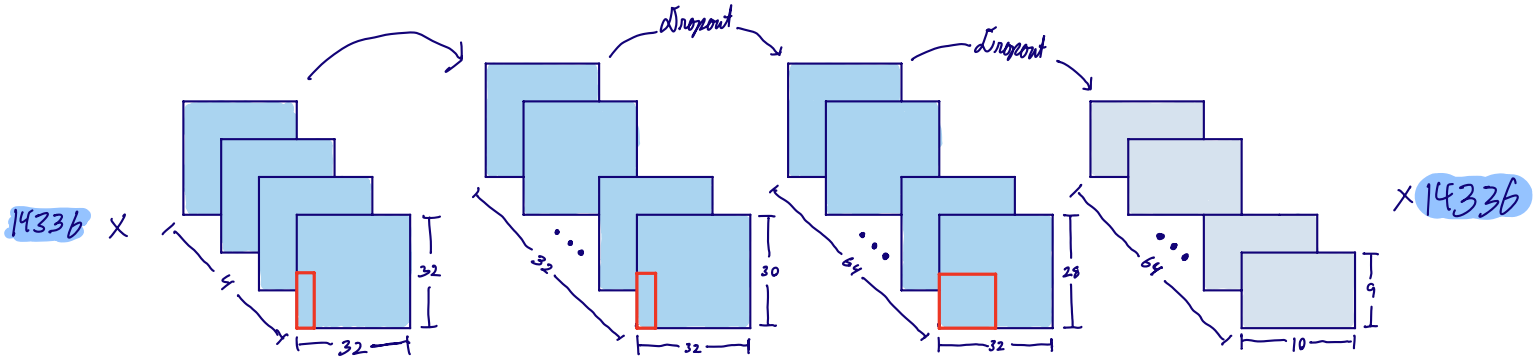
Max-Pooling Layer
- Pooling size = 3×3
- # of Kernels = 64

Stacking Layer

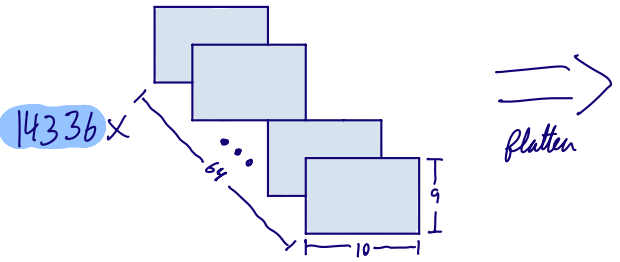
Output Layer

After Training, Drop Output Layer
- This makes the Stacking Layer the new Output Layer.

Note: Just using a CNN for classification is known to not have the best results. This is the main idea of the paper. We initially a 2-neuron output layer for training in order for back propagation in the CNN to adjust the weights, but then drop the output layer and take the data in the stacking layer, flatten it, and put it into the SAE.



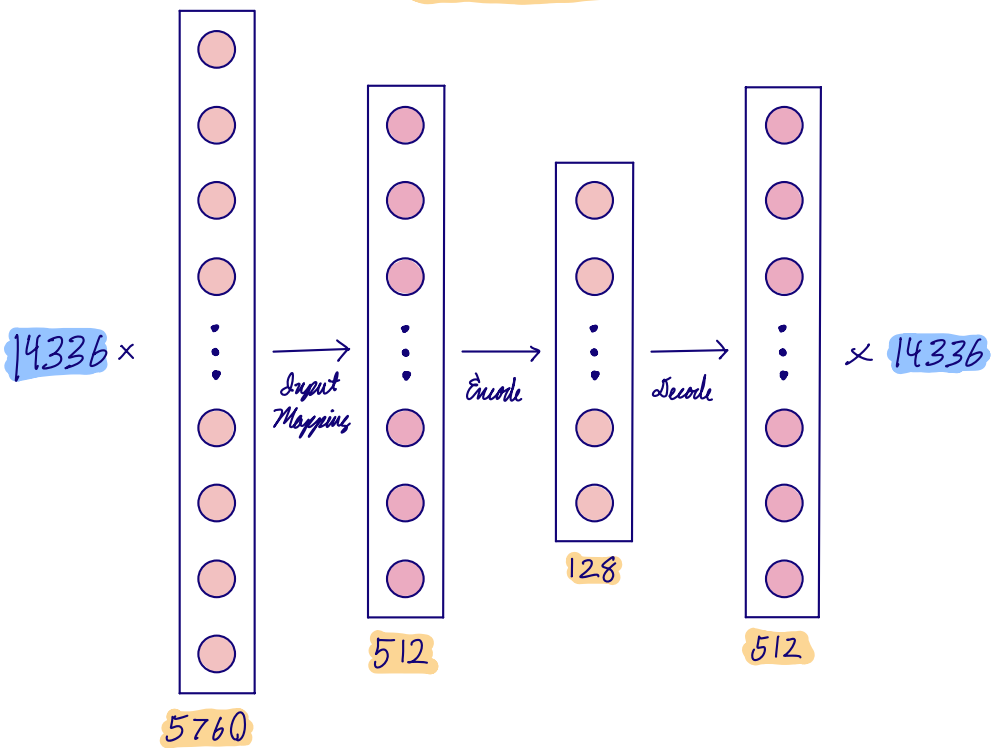
Flatten 3D output
From CNN into 1D Vector
- Data needs to be one-dimensional to go into SAE

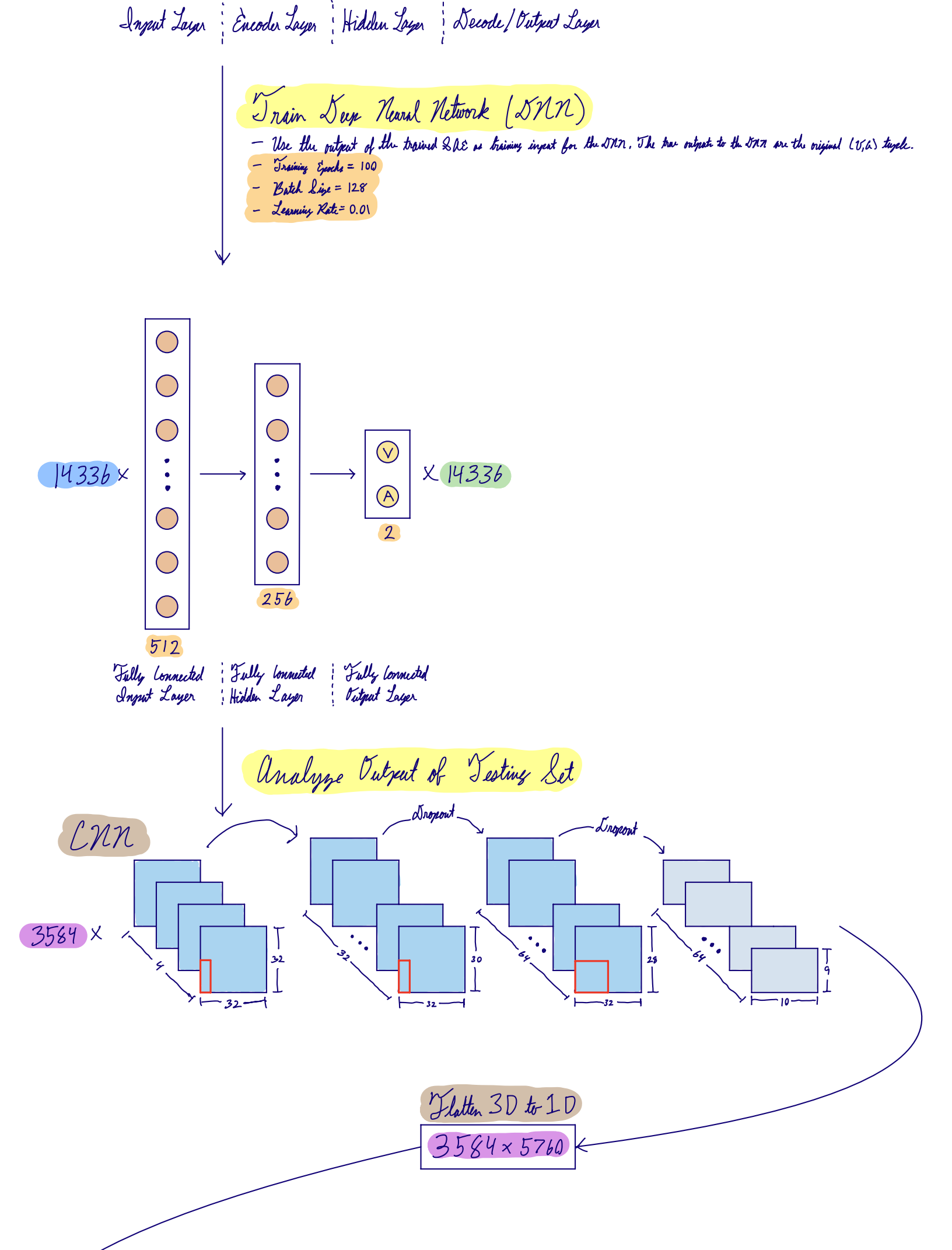


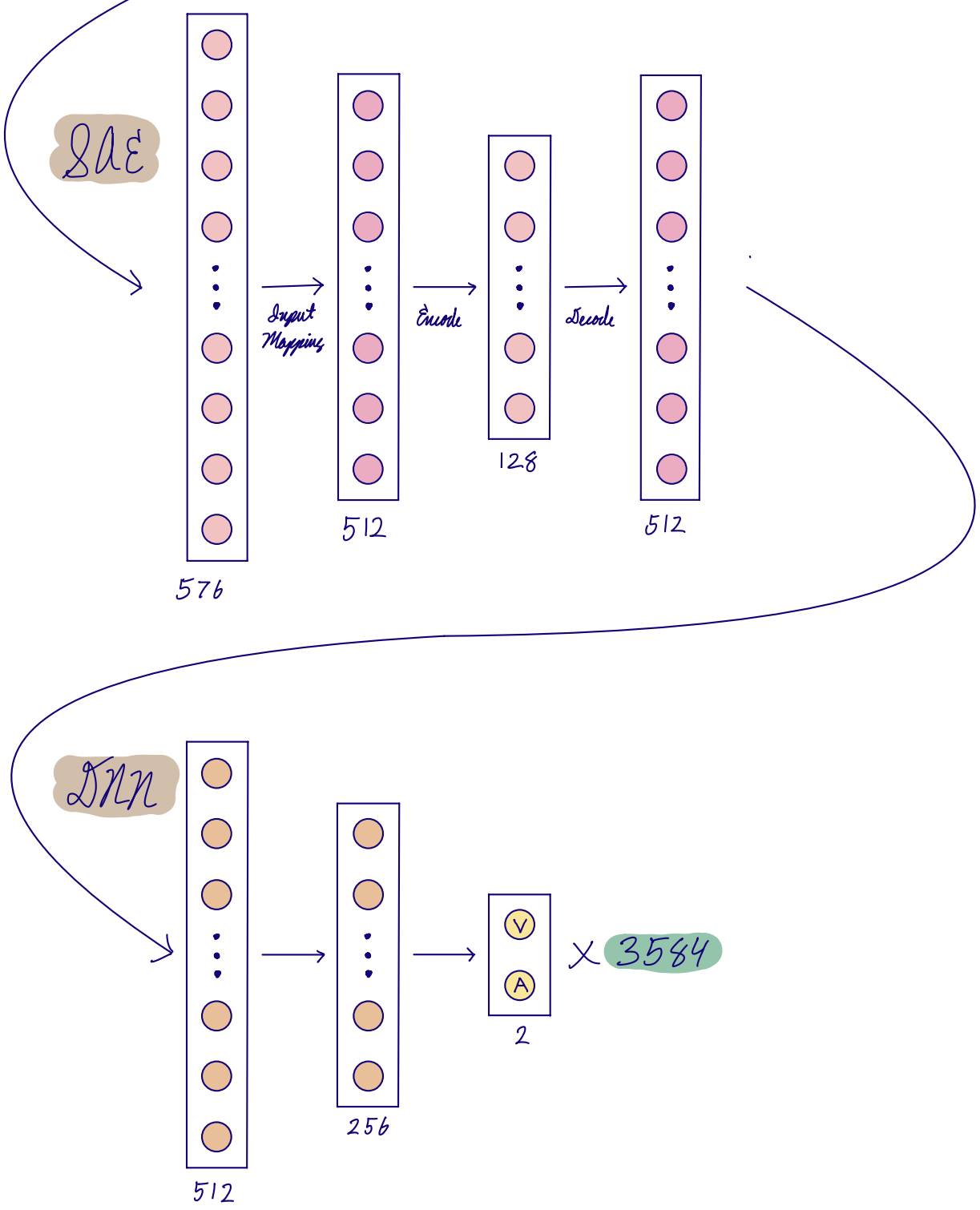
14336 x 5760 matrix

Train Sparse Autoencoder (SAE)

- SAE's are unsupervised learning models so we don't train it against an expected output
- Training Epochs = 100
- Batch Size = 64
- Learning Rate = 0.01




Input Layer
Encoder Layer
Hidden Layer
Decode / Output Layer
Train Deep Neural Network (DNN)
- Use the output of the trained SAE as training input for the DNN. The true outputs to the DNN are the original (V,A) tuples.
- Training Epochs = 100
- Batch Size = 128
- Learning Rate = 0.01
14336 x
512
256
V
A
2
x 14336
Fully Connected Input Layer
Fully Connected Hidden Layer
Fully Connected Output Layer
Analyze Output of Testing Set
CNN
Dropout
Dropout
3584 x
4
32
32
32
30
32
64
28
32
64
9
10
Flatten 3D to 1D
3584 x 5760


SAE
576
Input Mapping
512
Encode
128
Decode
512

DNN
512
256
V
A
2
X 3584