ENEL 525 Fall 2024 – Final Project

Nicholas Lam – 30115728

Dec 18, 2024

University of Calgary

Table of Contents

Introduction

This project involves developing a convolutional neural network (CNN) to classify aerial satellite images based on their land use categories. The goal is to utilize deep learning techniques, extract relevant features, and achieve accurate classification of various land types. By going through the exercise of identification and recognition of different patterns in the dataset, I can learn about the effectiveness of CNNs in handling image classification and how they work.

GitHub repo link: https://github.com/Nicholas-Lam1/ENEL525_Project.git

Methodology

I have been given a dataset containing 2100 aerial satellite images, consisting of 21 different land use categories. To help identify these land use categories, I require a neural network model which can identify new input images to a high degree of accuracy.

I will be utilizing Python with the TensorFlow library to develop a CNN model to identify the land use category of input images. From the 2100 images, training will utilize 80% of the images and the last 20% will be used for validation/testing (2/3 for validation, 1/3 for testing).

Model Creation Method

The initial model is based off of the one given in Exercise 3 for the final project [1]. During testing, each change to a parameter was run through three different shuffling seeds, 1, 12, and 123. These changes were then compared based on the results of each seed, and the best value for the parameter was selected. The order in which different elements of the model were finalized were as follows:

1. Number of Conv2D layers
2. Number of Dense layers
3. Learning rate
4. Number of Dropout layers
5. Dropout percentage
6. Number of Epochs (based on EarlyStopping)

After these parameters were set, features such as padding and increased kernel size were tested, but found to not have much impact. As such, they were left as they were originally.

Please refer to the images in the GitHub repository to see further information on testing. I was not able to compile all my results to a table due to improper recording of results. However, the images still reflect my process of testing to create the model.

Final Network/Model

Based on the methodology mentioned above, the following model was created. It has four convolution layers using the rectified linear unit (ReLU) activation function, each with a pooling layer between them. Following the convolution layer, there are two hidden dense layers, also using the ReLU activation function. Lastly, the classification dense layer, using the softmax activation function. Dropouts are used between every layer after the first two convolution layers.

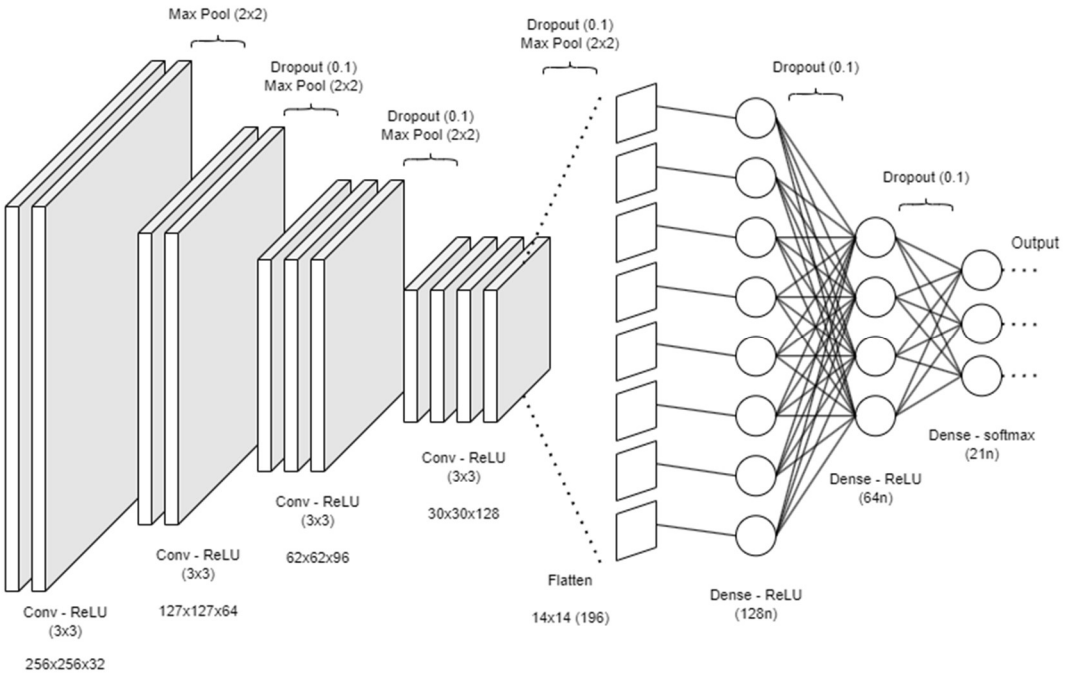

FIGURE 1: NETWORK DIAGRAM

```
model = Sequential()

model.add(Conv2D(32, (3, 3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2, 2)))

model.add(Conv2D(64, (3, 3), activation = 'relu'))
model.add(Dropout(0.1))
model.add(MaxPooling2D(pool_size = (2, 2)))

model.add(Conv2D(96, (3, 3), activation = 'relu'))
model.add(Dropout(0.1))
model.add(MaxPooling2D(pool_size = (2, 2)))

model.add(Conv2D(128, (3, 3), activation = 'relu'))
model.add(Dropout(0.1))
model.add(MaxPooling2D(pool_size = (2, 2)))

model.add(Flatten())

model.add(Dense(128, activation = 'relu'))
model.add(Dropout(0.1))
model.add(Dense(64, activation = 'relu'))
model.add(Dropout(0.1))

model.add(Dense(21, activation = 'softmax'))

model.compile(
    optimizer = tf.keras.optimizers.Adam(0.0005),
    loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits = False),
    metrics = [tf.keras.metrics.SparseCategoricalAccuracy()],
)

early_stopping = EarlyStopping(
    monitor = 'val_loss',
    patience = 3,
    restore_best_weights = True
)

history = model.fit(
    ds_train,
    epochs = 25,
    validation_data = ds_val,
    callbacks = [early_stopping],
)

model.summary()
```

FIGURE 2: MODEL STRUCTURE IN CODE

Parameters

The following are explanations for why each parameter was selected, the selection process follows what was described in the Model Creation Method section.

Learning Rate: 0.0005

Learning rates from 0.001 to 0.0001 were tested, it was found that 0.001 led to poor training which was quick but very inconsistent in the resulting accuracy and loss. As the learning rate was tuned down, 0.0005 seemed to be a sweet spot where accuracy was consistently above 90% with a loss of less than 0.3 in training and 0.5 in validation. These results seemed to be the maximum performance from the model, being the same at rates lower than 0.0005, as a result, the learning rate was raised again to 0.0005.

Batch Size: 100

Batch size was fixed arbitrarily, other parameters were used to tune training stability and computation speed.

Number of Epochs: 25

The number of epochs was determined by when the learning rate plateaued. The model was left running with 50 epochs and an early stopping mechanism when the rate of loss stagnated for 3 epochs in a row. It was found that this model stagnated at around 23 epochs, as such, the number of epochs was set to 25.

Optimizer: Adam

The Adam optimizer was selected as it was recommended from class and the easiest to manipulate using the parameters. It seemed like a good default choice which would have good application to this model.

Dropout: 0.1

The value of the dropouts was determined through testing after the learning rate was set. Values from 0.4 to 0.01 were tested, with lower values in the range of 0.05 to 0.15 giving the smoothest and consistent accuracy and loss curves. High values often caused the model to take longer to train, while giving similar results, while values below 0.05 caused the models testing accuracy to be significantly lower than the training accuracy (overfitting). A value of 0.1 was settled on, as it consistently allowed the model to reach 90%+ accuracy in around 25 epochs.

Kernel Size: 3x3

The kernel size of 3x3 was first selected as it was used in the exercises prior to the project. 5x5 and 7x7 was tested after the learning rate and dropouts were finalized, however, they did not perform as well as 3x3, so 3x3 was left.

Activation Function: ReLU

Similar to the optimizer, the ReLU activation function was selected as it was used in class. Leaky ReLU was looked at after all testing was completed, but it's primary benefit, the prevention of dead neurons, was mitigated with the use of dropout, making it unfavorable to attempt to change.

Activation Function: Softmax

Softmax was required to get an output for multi-class classification.

Padding: None

Padding is not used, as most of the dataset had features on the edges of the image. The airplane, sparse residential, intersection, and baseball diamond categories were the only categories which contained their distinctive features primarily in the center of the image.

Datasets:

The training dataset utilized 80% of the images and the last 20% was used for validation/testing (2/3 for validation, 1/3 for testing).

Backpropagation (For ReLU Neurons)

Backpropagation in a convolutional neural network (CNN) works by updating weights to minimize the loss function. The process involves two main steps, forward propagation, where the input is passed through the network to predict classification, and backward propagation, where the loss is used to adjust the weights.

During forward propagation, the dot product (Frobenius) of each 3x3 segment of the image (or previous layer feature map) and the kernel is used to form a feature map within each convolution layer. Once at the hidden layers, this feature map is multiplied by weight and bias is added. The result is passed to the ReLU activation function.

$$ReLU(x) = \max(0, x) \begin{cases} x \; if \; x > 0 \\ 0 \; if \leq 0 \end{cases}$$

EQUATION 1: RECTIFIED LINEAR UNIT ACTIVATION FUNCTION

During backpropagation, the sensitivities are calculated using the following.

$$F = \frac{d}{dx} ReLU(x) = \begin{cases} 1 \; if \; x \geq 0 \\ 0 \; if \; x < 0 \end{cases}$$

EQUATION 2: RECTIFIED LINEAR UNIT ACTIVATION FUNCTION DERIVATION

$$\mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t} - \mathbf{a}), \qquad \mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T \mathbf{s}^{m+1}, \text{ for } m = M-1, \ldots, 2, 1.$$

EQUATION 3: SENSITIVITY UPDATE CALCULATIONS [8]

After the sensitivities have been calculated, the weight and bias can be updated using the following equations.

$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha \mathbf{s}^m(\mathbf{a}^{m-1})^T,$$

$$\mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha \mathbf{s}^m.$$

EQUATION 3: WEIGHT AND BIAS UPDATE CALCULATIONS [8]

Loss Function

The loss function of cross-entropy is as follows:

$$Loss = -\sum_{i=1}^{n} t_i \cdot \log(p_i), for\ n\ amount\ of\ classes$$

EQUATION 4: CATEGORICAL CROSS-ENTROPY EQUATION

Where t_i is the true/actual label of the image, and p_i is the predicted classification of the image from the softmax activation function.

Results and Discussion

In Figure 3, the accuracy and learning curve for both validation and testing is shown. The training takes place over 25 epochs, and as shown in Figure 4, the model results in a training accuracy of 95.89% and validation accuracy of 92.00%. There is a degree of overfitting, however, with our limited dataset for training, this is an acceptable result after 25 epochs.

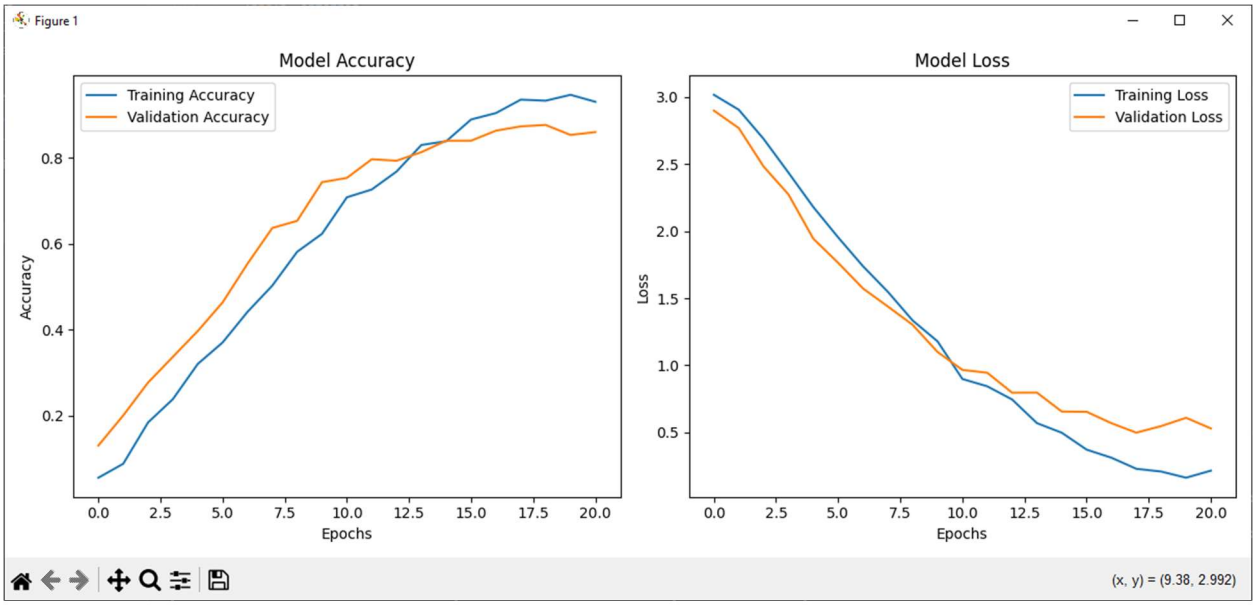

FIGURE 4: ACCURACY AND LOSS CURVES

```
Epoch 15/25
17/17 ━━━━━━━━━━━━━━━━━━━━ 45s 3s/step - loss: 0.4889 - sparse_categorical_accuracy: 0.8376 - val_loss: 0.5507 - val_sparse_categorical_accuracy: 0.8967
Epoch 16/25
17/17 ━━━━━━━━━━━━━━━━━━━━ 45s 3s/step - loss: 0.4260 - sparse_categorical_accuracy: 0.8607 - val_loss: 0.4838 - val_sparse_categorical_accuracy: 0.9033
Epoch 17/25
17/17 ━━━━━━━━━━━━━━━━━━━━ 44s 3s/step - loss: 0.3606 - sparse_categorical_accuracy: 0.8998 - val_loss: 0.4944 - val_sparse_categorical_accuracy: 0.9000
Epoch 18/25
17/17 ━━━━━━━━━━━━━━━━━━━━ 44s 3s/step - loss: 0.2792 - sparse_categorical_accuracy: 0.9143 - val_loss: 0.4643 - val_sparse_categorical_accuracy: 0.9233
Epoch 19/25
17/17 ━━━━━━━━━━━━━━━━━━━━ 45s 3s/step - loss: 0.2261 - sparse_categorical_accuracy: 0.9374 - val_loss: 0.4050 - val_sparse_categorical_accuracy: 0.9400
Epoch 20/25
17/17 ━━━━━━━━━━━━━━━━━━━━ 44s 3s/step - loss: 0.2165 - sparse_categorical_accuracy: 0.9364 - val_loss: 0.4593 - val_sparse_categorical_accuracy: 0.9267
Epoch 21/25
17/17 ━━━━━━━━━━━━━━━━━━━━ 44s 3s/step - loss: 0.2197 - sparse_categorical_accuracy: 0.9348 - val_loss: 0.3979 - val_sparse_categorical_accuracy: 0.9267
Epoch 22/25
17/17 ━━━━━━━━━━━━━━━━━━━━ 44s 3s/step - loss: 0.1857 - sparse_categorical_accuracy: 0.9429 - val_loss: 0.3998 - val_sparse_categorical_accuracy: 0.9133
Epoch 23/25
17/17 ━━━━━━━━━━━━━━━━━━━━ 44s 3s/step - loss: 0.1772 - sparse_categorical_accuracy: 0.9487 - val_loss: 0.3913 - val_sparse_categorical_accuracy: 0.9300
Epoch 24/25
17/17 ━━━━━━━━━━━━━━━━━━━━ 45s 3s/step - loss: 0.1350 - sparse_categorical_accuracy: 0.9588 - val_loss: 0.3937 - val_sparse_categorical_accuracy: 0.9367
Epoch 25/25
17/17 ━━━━━━━━━━━━━━━━━━━━ 46s 3s/step - loss: 0.1262 - sparse_categorical_accuracy: 0.9589 - val_loss: 0.4813 - val_sparse_categorical_accuracy: 0.9200
```

FIGURE 3: EPOCH (10-25) RESULTS

The full image of epochs 3-25 is provided at the end of the document, epochs 1 and 2 are excluded due to the limits of the terminal size, it would not display all 25 epochs.

Model Layers

The CNN model consists of:

- 4 Conv2D layers (3x3 kernels) with pooling (2x2), increasing the number of filters per layer from 32 to 128.
- Dropout layers after the last three convolutional layers to reduce overfitting.
- A fully connected network with Dense layers (128 and 64 neurons) followed by a Softmax output layer for classification.

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 254, 254, 32)	896
max_pooling2d (MaxPooling2D)	(None, 127, 127, 32)	0
conv2d_1 (Conv2D)	(None, 125, 125, 64)	18,496
dropout (Dropout)	(None, 125, 125, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 62, 62, 64)	0
conv2d_2 (Conv2D)	(None, 60, 60, 96)	55,392
dropout_1 (Dropout)	(None, 60, 60, 96)	0
max_pooling2d_2 (MaxPooling2D)	(None, 30, 30, 96)	0
conv2d_3 (Conv2D)	(None, 28, 28, 128)	110,720
dropout_2 (Dropout)	(None, 28, 28, 128)	0
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 128)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 128)	3,211,392
dropout_3 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 64)	8,256
dropout_4 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 21)	1,365

Total params: 10,219,553 (38.98 MB)
Trainable params: 3,406,517 (12.99 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 6,813,036 (25.99 MB)

FIGURE 5: MODEL LAYERS

Testing Results

Here are the results of the testing prediction done by the model. The number of images tested varies, as the testing set was taken as a subset of the initial scrambled dataset.

Predicted / Expected	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	**9**	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	**21**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	**9**	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	**15**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	**20**	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1
6	1	0	0	0	0	**15**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	1	0	**9**	0	1	0	0	0	0	2	0	0	0	0	0	0	0
8	2	0	0	0	0	0	0	**8**	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	1	0	0	2	0	0	0	**5**	0	0	0	0	0	0	0	1	0	0	0	0
10	0	0	2	0	0	0	0	0	0	**9**	0	1	0	0	0	0	0	0	1	0	0
11	0	0	0	0	0	0	0	0	0	0	**18**	0	0	2	0	0	0	0	0	0	0
12	0	0	0	0	0	0	1	0	0	0	0	**11**	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	**9**	0	1	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	1	**20**	0	1	0	0	0	0	0
15	0	0	0	0	1	0	0	0	0	0	0	0	0	0	**11**	0	0	0	0	0	0
16	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	**17**	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**7**	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	**9**	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**14**	1	0
20	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**13**	1
21	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	**13**
Total	13	23	12	16	26	16	13	8	6	9	18	12	10	24	13	18	9	9	15	15	15
Accuracy	69%	91%	75%	94%	77%	94%	69%	100%	83%	100%	100%	92%	90%	83%	85%	94%	78%	100%	93%	87%	87%
Avg. Accuracy	87.67%																				

TABLE 1: CONFUSION MATRIX AND ACCURACY

Results Discussion

The model performed quite similarly to how it did during training and validation, with an avg. accuracy of 87.67%, compared to the 92.00% accuracy from validation. The two classes which performed poorly, agricultural (set 1) and dense residential (set 7), presented unique challenges for feature extraction. The agricultural set provides very little detail which can be extracted by the model, while the dense residential set contains too much variety of detail, with different coloured roofs, varying street layouts, while also looking like the medium residential set.

I believe this model would be able to accurately predict any images from outside the dataset, as the dataset already contained a good amount of variation (rotation, variety of buildings, and colour). However, I believe that its accuracy is limited to around 85%. The training was quite robust, but as shown in Figure 3 and 4, the peak it can reach with its own dataset is ~92%.

Conclusion

Despite a small degree of overfitting, the model demonstrated robust training and generalization capabilities. The performance on testing data aligns closely with validation accuracy, suggesting the model could achieve approximately 85% accuracy on new images from outside the dataset.

Although fairly accurate, there is a lot of room for improvement. Some main improvements which I wish I could make are to test stride distance in the convolution layers and to try and look at pre-trained models. I believe stride distance would help immensely in compute time, as the detail across the images is quite even, meaning that larger strides can be taken without harming the feature map of the model. Pre-trained models also would help reduce compute time and perhaps serve as a more solid foundation for the model.

Additionally, outside of the model, I had tested doubling the dataset by rotating the images to help increase the amount of training data. Unfortunately, I was not able to get it to work. This change could potentially help make the model more robust/accurate, and help provide a larger dataset not just for training, but also for validation.

The project highlights CNNs' potential for feature extraction and classification tasks, even with constrained datasets.

References

[1] TensorFlow, "Image classification," *TensorFlow Tutorials*, [Online]. Available: https://www.tensorflow.org/tutorials/images/classification?_gl=1*zh0l1c*_up*MQ..*_ga*NTUwNjAzMjQ4LjE3MzQzMTIwMzM.*_ga_W0YLR4190T*MTczNDMxMjAzMi4xLjAuMTczNDMxMjAzMi4wLjAuMA. [Accessed: Dec. 9, 2024].

[2] Keras, "Layers API," *Keras Documentation*, [Online]. Available: https://keras.io/api/layers/. [Accessed: Dec. 9, 2024].

[3] TensorFlow, "Overfitting and underfitting," *TensorFlow Tutorials*, [Online]. Available: https://www.tensorflow.org/tutorials/keras/overfit_and_underfit. [Accessed: Dec. 9, 2024].

[4] "How to avoid overfitting in CNN," *Stack Overflow*, [Online]. Available: https://stackoverflow.com/questions/62136364/how-to-avoid-overfitting-in-cnn. [Accessed: Dec. 10, 2024].

[5] "Trouble with EarlyStopping in tf.keras," *Stack Overflow*, [Online]. Available: https://stackoverflow.com/questions/60627667/trouble-with-earlystopping-in-tf-keras. [Accessed: Dec. 12, 2024].

[6] V. Patel, "Gentle dive into math behind convolutional neural networks," *Towards Data Science*, Mar. 22, 2020. [Online]. Available: https://towardsdatascience.com/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9. [Accessed: Dec. 15, 2024].

[7] P. Viswanathan, "Convolutions and backpropagations," *Medium*, Oct. 17, 2020. [Online]. Available: https://pavisj.medium.com/convolutions-and-backpropagations-46026a8f5d2c. [Accessed: Dec. 18, 2024].

[8] M. T. Hagan, H. B. Demuth, and M. H. Beale, *Neural Network Design*, 2nd ed. Boston, MA: PWS Publishing, 1996.

Appendix

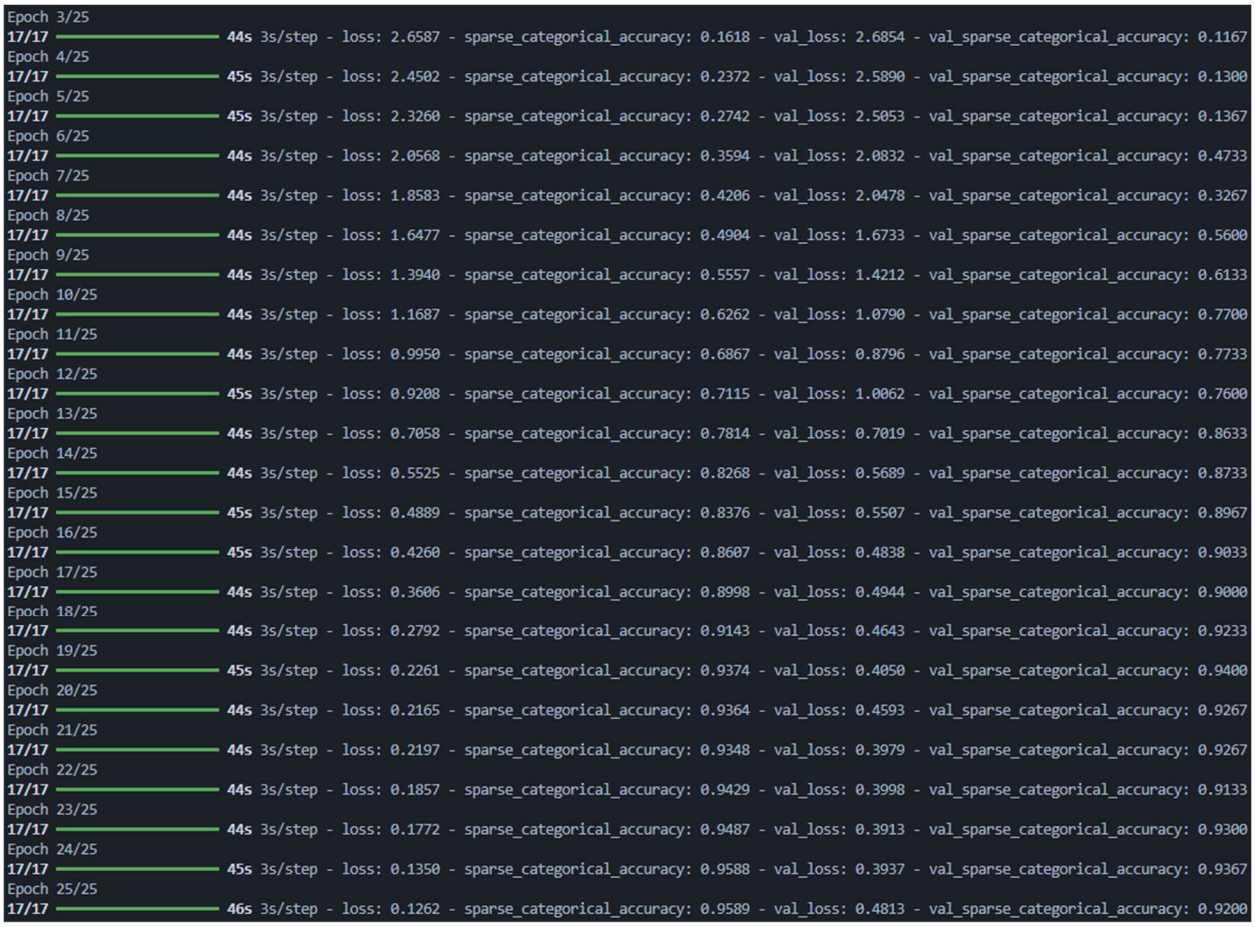

FIGURE 6: EPOCH (3-25) RESULTS