Deep Learning
Project Documentation

Wali, Ahmad Mustapha (Gr. 507, Artificial Intelligence)

Project Summary

This is project medical image classification competition in which train deep classification models trained on magnetic resonance images (MRI) from a data set. Each MRI has 3 unnamed labels, each of which is binary, i.e., each image can belong to multiple classes. For this project, the mean F1 score on a given test set is used as evaluation metric. The mean F1 score is the average F1 score of the labels.

2 convolutional neural network (CNN) models were used for the project; an AlexNet-type model, and a VGG-type model. The models were trained with different iterations of the same data, which is sometimes augmented to provide for more generalization. The models' predictions were also post-processed by adjusting a threshold value to give optimal performance.

Data

The datasets provided comprised 12000 training images, 3000 evaluation images, and 5000 test images. Each of the training and validation images had a label, i.e., no missing values.

A little over 25% (3092) of the train images were found to be completely black. This figure was obtained by counting all images in the training set that have a mean of 0 *(image.mean() == 0)*.

There were also some observations regarding class imbalance in the labels, particularly Label 3 and Label 2.

	Label 1	Label 2	Label 3
0	53.63%	63.88%	73.33%
1	46.37%	36.12%	22.67%

Table 1 Class distribution by label

Models

2 models were used for the task:

i. An AlexNet-type model with 5 convolutional blocks, and 1 output block. The first convolutional block had a convolutional layer of 32 5x5 filters, a maxpooling layer of size 4x4 and a stride of 2 and padding, a dropout layer with a rate of 25%. Each of the parameters of the convolutional blocks remain the same besides the number of filters, which were geometrically increased by a factor of 2 to make the final layer have 512 filters. The output block starts with a flattening layer, followed by a dense layer of 1024 nodes, a dropout layer with a rate of 50% and an output layer of 3 nodes with a sigmoid activation function.

ii. A VGG-type model with 5 convolutional blocks and 1 output block. The first convolutional block had a first convolutional layer of 32 5x1 filters, a second convolutional layer of 32 1x5 filters, a maxpooling layer of size 3x3 and a stride of 2 and padding, a dropout layer with a rate of 25%. Each of the parameters of the convolutional blocks remain the same besides the number of filters, which geometrically increases by a factor of 2 to make the final layer have 256 filters. The output block starts with a flattening layer, followed by a dense layer of 512 nodes, a dropout layer with a rate of 50% and an output layer of 3 nodes with a sigmoid activation function.

Enhancing Generalization

To enhance generalization several methods were used:

i. **Preprocessing**: The data was preprocessed by scaling it by 4 different factors. As they are images with pixel values between 0-255, a model's performance can greatly improve by scaling the image data. For this the data was scaled to one of either:

I. 'none': the raw images are used as features without any preprocessing.

II. 'minmax': An implementation of the *minmax* scaler, which scales all pixel values to a range of float values between 0 – 1 from 0-255. This greatly improves performance.

$$X = \frac{x - \min(X)}{\max(X) - \min(X)}$$

III. 'normalized': Cantering the mean of the whole dataset around 0. This is particularly good for normally-distributed data.

$$X = \frac{x - \min(X)}{\max(X) - \min(X)} - 0.5$$

IV. ‘standard’: An implementation of a *standardscaler*, where the mean of the dataset is made to 0, and the variance made to be 1.

$$X = \frac{x - X.mean()}{X.std()}$$

ii. Dropout regularization: Using dropout regularization to reduce model complexity, which in turn reduces overfitting. Every convolutional block has a dropout rate of 25%, while the output has dropouts of 50%.

iii. Weight initialization: Several weight initialization techniques were tried, but one that gave the best results was *he_uniform*. This method initialized a layer’s weights with uniformly-distributed random numbers multiplied by a factor of the previous layer’s dimensions.

$$np.random.rand() * \sqrt{\frac{2}{Previous\ layer's\ dimensions}}$$

iv. Data augmentation

A data augmentation strategy was implemented on only the training data using Keras’s ImageDataGenerator. The images were saved to locally so that they can be preprocessed together with the rest of the dataset. The labels of the generated images were added to the original training labels. The augmented image size that gave optimum results was with 3000 images.

v. **Callbacks**

I. Early stopping and learning rate reduction were used as callbacks to also reduce overfitting. The models stop training after 5 of the validation loss not improving.

II. Keras’s ReduceLROnPlateau reduces the learning rate by a factor of 0.1 after 2 epochs of the validation loss not improving.

Metrics

The metric closest to mean F1 score in Keras was the area under curve (AUC) as it allows for multilabel classification. The AUC measures the area under the ROC curve, which is a graph of the true positive rates against the false positive rates at different thresholds. The true positive rate is given as

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

And the false positive rate (FPR) is given as

$$FPR = \frac{FP}{FP + TN}$$

An ideal value for the AUC is 1, when the TPR = 1, and FPR = 0.

Post processing

With the output layers of the models being sigmoid, they return an array of floats with ranges between 0 – 1. The float values were rounded up or down relative to a set threshold. This method has produced the best results. In a blogpost, Honda (2021) proved that to maximize a model's F1 score, the threshold should be set to

$$s \geq \frac{F}{2}$$

where s is the threshold, and F is the maximum attainable F1 score for the task (which, in this case is about 0.95)

Results and graphs

The results obtained for both models are below

Scale		"none"	"minmax"	"standard"	"normalized"
Model 1	Training AUC	0.8843	0.8951	0.8955	0.8202
	Validation AUC	0.9005	0.9059	0.8898	0.7510
Model 2	Training AUC	0.7762	0.8792	0.8594	0.7867
	Validation AUC	0.754	0.9096	0.8964	0.7954

Table 2 Performance Results with Augmentation

Scale		"none"	"minmax"	"standard"	"normalized"
Model 1	Training AUC	0.9352	0.9455	0.9473	0.8975
	Validation AUC	0.9006	0.9039	0.8979	0.8158
Model 2	Training AUC	0.8969	0.9358	0.9195	0.8401
	Validation AUC	0.8915	0.9015	0.9031	0.7712

Table 3 Performance Results Without Augmentation

Per class Precision		Label 1	Label 2	Label 3
Model 1	Class 0	0.91	0.92	0.95
	Class 1	0.91	0.93	0.88
Model 2	Class 0	0.91	0.92	0.95
	Class 1	0.91	0.87	0.83

Table 4 Per-Class Precision Without Augmentation for Best-Performing Models

Per class Precision		Label 1	Label 2	Label 3
Model 1	Class 0	0.95	0.93	0.95
	Class 1	0.89	0.84	0.78
Model 2	Class 0	0.96	0.96	0.96
	Class 1	0.88	0.80	0.65

Table 5 Per Class Precision with Augmentation for Best-Performing Models

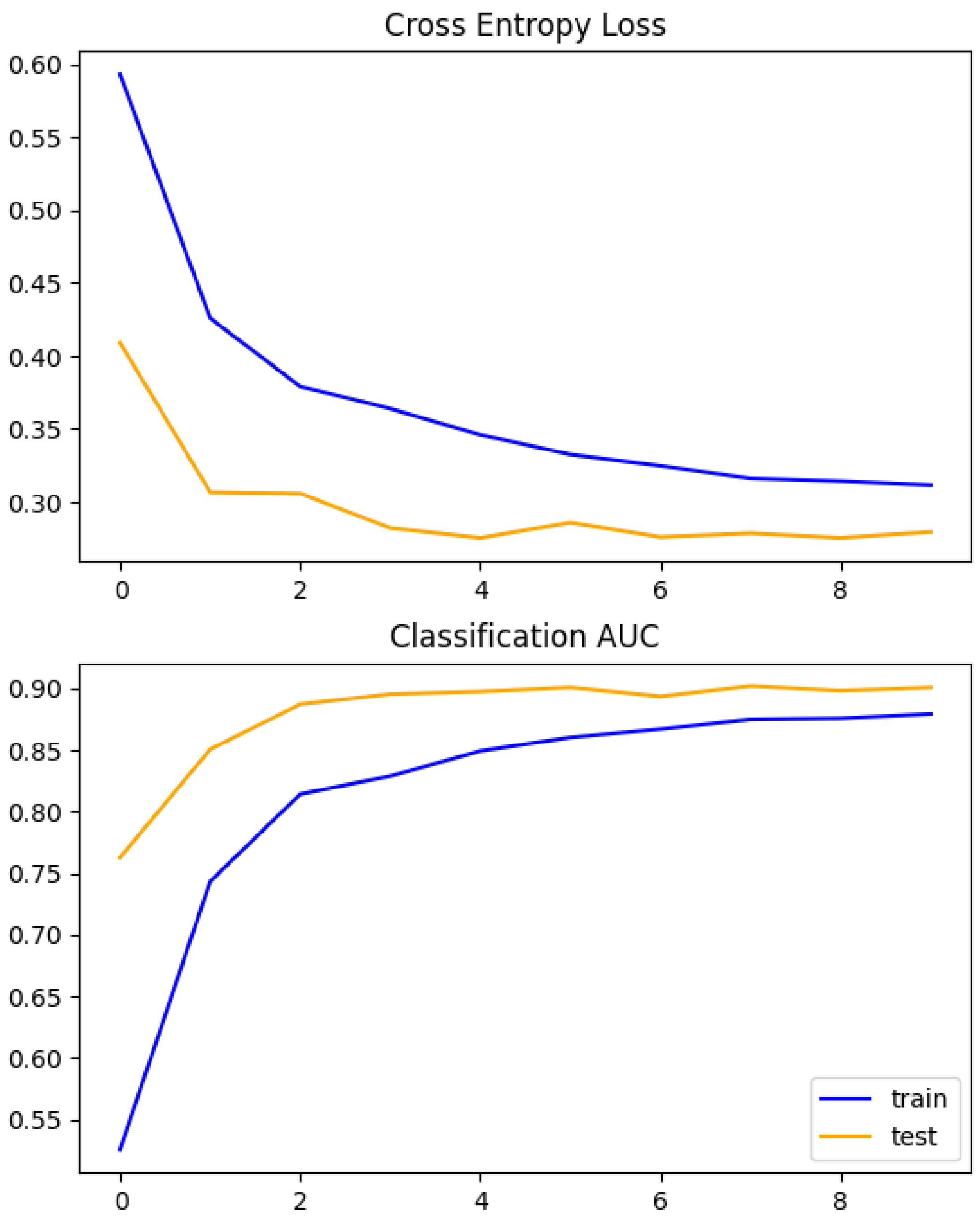

Figure 1 Best Performing Model 2 with Augmentation

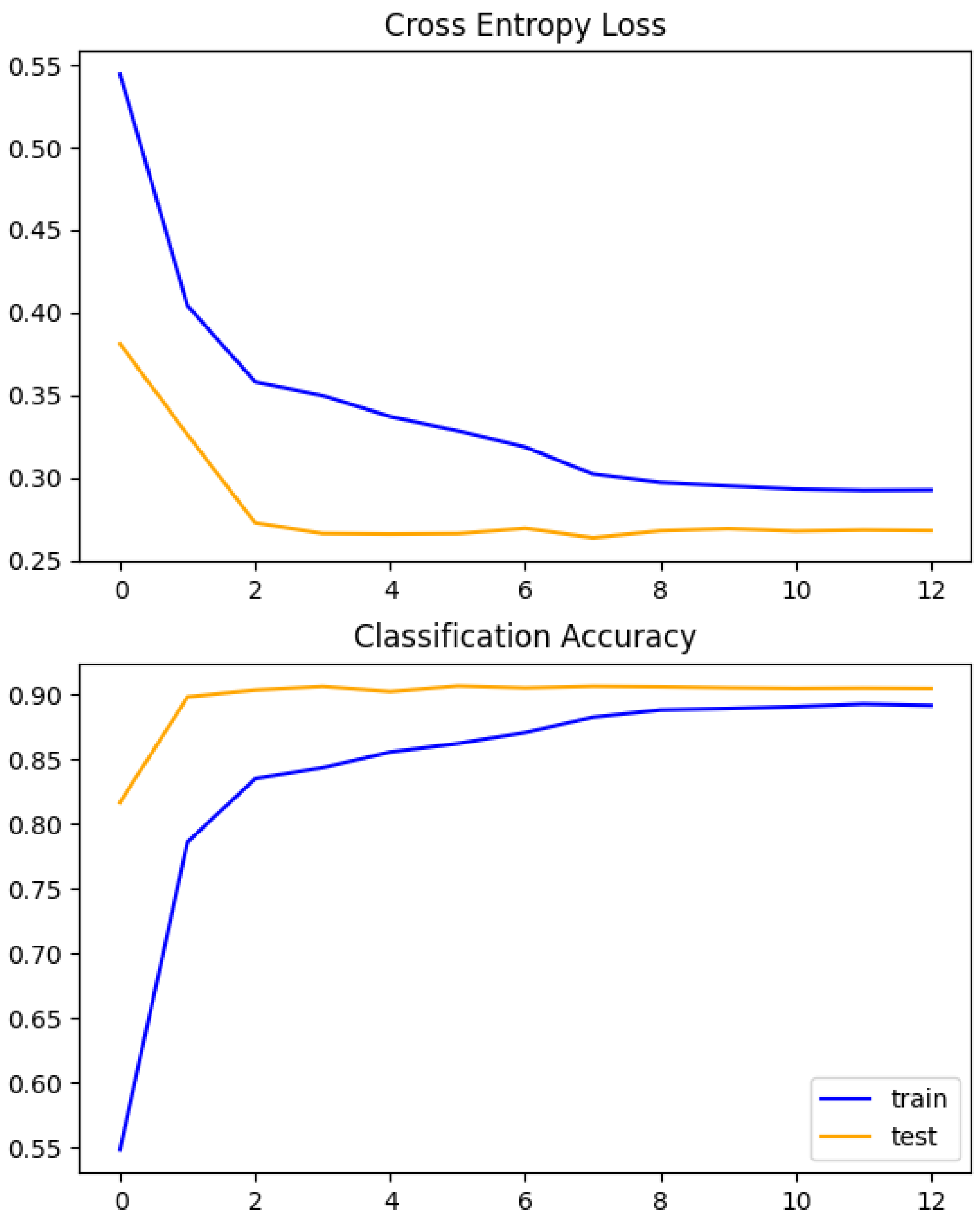

Figure 2 Best Performing Model 1 with Augmentation

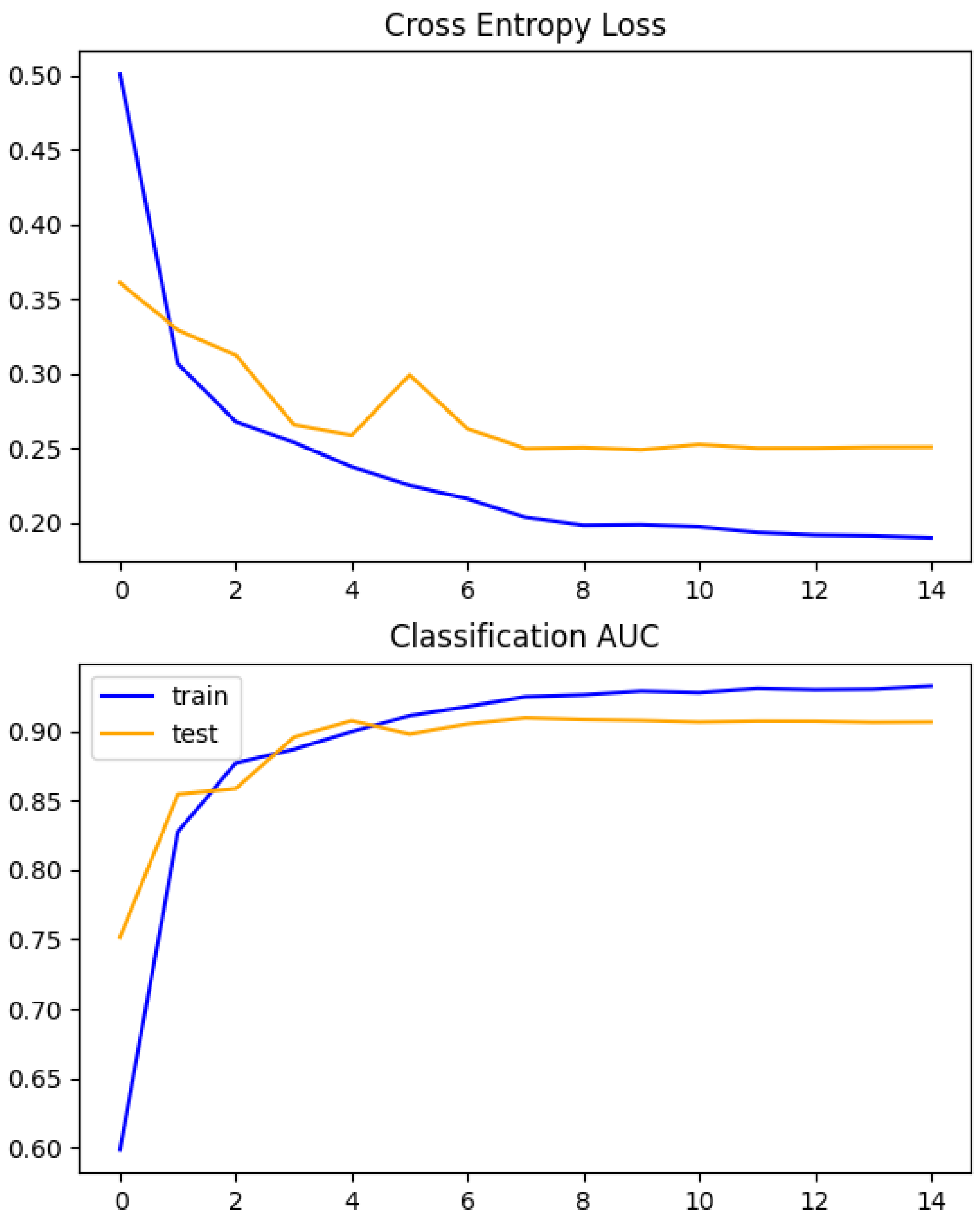

Figure 3 Best Performing Model 1 without Augmentation

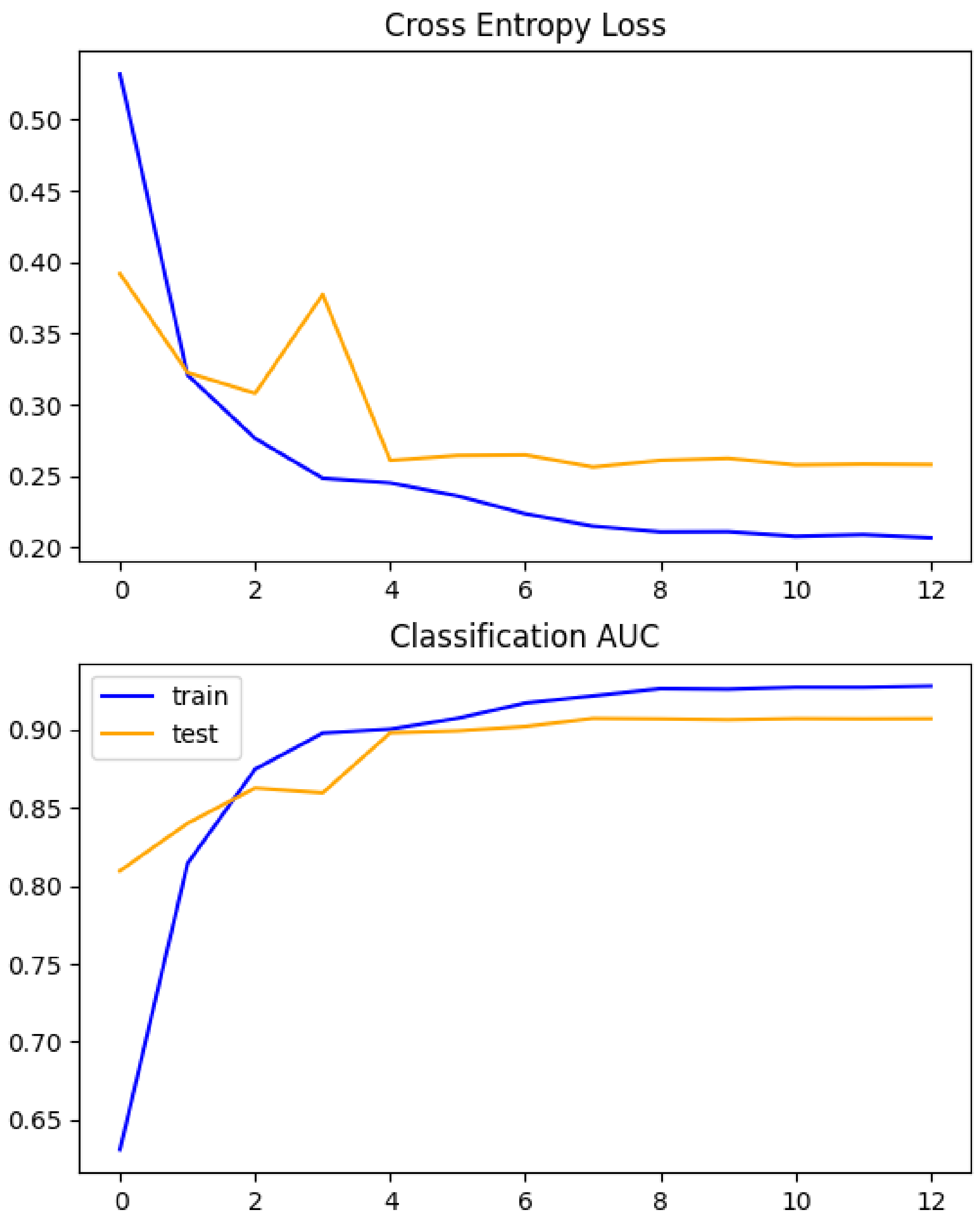

Figure 4 Performing Model 2 without Augmentation