Evaluating Deep Learning Approaches for Out-of-Domain Land Cover Classification

Matthew Kolodner Jack Xiao
Stanford University
{mkolod, jackxiao}@stanford.edu

Abstract

Training a classifier that performs well on out-of-domain data is often challenging, especially if the out-of-domain data has a significantly different distribution from the in-domain training data. One problem where training an effective out-of-domain classifier would be applicable is the mapping of land cover in low-income regions based on low-resolution satellite imagery. With access to a high volume of well labeled land cover data from various regions around the world and a smaller set of labeled data specific to sub-Saharan Africa, we aim to explore the ability of 3 different deep learning approaches to generalize well on out-of-domain test data: a Convolutional Neural Network, a Long Short-Term Memory network, and a Transformer Model. We find that deep learning models achieve some success over a K-Nearest Neighbors baseline model, with the LSTM model outperforming the others. We also investigate how performance changes on out-of-domain data compared to in-domain data when several deep learning changes are to normalization methods, dropout values, and batch sizes. We hope these experiments can inform decisions for others regarding general approaches to out-of-domain tasks.

1. Introduction

1.1. Motivation

Understanding changes to Earth's surface is one of the keys to combating the loss of natural habitats and fertile land. Having a way to accurately map land cover around the globe and monitor changes would be a valuable tool in the preservation of our planet. The primary challenge in creating such a model is the lack of viable land-cover data in low-income regions. There is, however, plenty of accurate land cover data from higher-income regions, which suggests taking an approach that leverages the information we do have (despite it being from completely different regions) to have a more consistent model. We will explore methods to more accurately classify land cover given low-resolution satellite data with minimal data from target regions. Not only would having the ability to correctly classify land cover in developing regions be useful in informing policy decisions to achieve sustainability goals, but the results of our generalization experiments can potentially be applicable to other classification problems.

1.2. Problem

The land cover data from lower-income regions are limited in both quantity and resolution. There are not enough accurately labeled data points from areas such as sub-Saharan Africa to train a good classifier. Developing a classifier for land cover in that region will require the use of the more abundant data from higher-income areas and other out-of-domain techniques. In addition, the data that is available for those regions is primarily low-resolution satellite imagery that does not lend itself well to training effective land cover classifiers due to the large area aggregated by each individual pixel and the lack of finer details that might differentiate between land cover types. The data that is available is much more suitable for classification method based on individual pixels from the satellite data rather than one based on full images.

1.3. Approach

In this paper, we evaluate the ability of various deep learning models to accurately classify land cover on both in-domain and out-of-domain data points. The input is a 7 x 46 time series of a satellite image pixel representing 7 channels extracted over 46 days, which we will run through a classifier to obtain 1 of 17 land cover labels. In this paper, we use four different models (a KNN baseline, a CNN model, and LSTM model, and a Transformer model) on the input to see which one results in the highest performance. We also hope that the results of our experiments can be useful to inform solutions and model designs for other out-of-domain tasks.

2. Related Work

There have been promising results in developing techniques for few-shot land cover classification. A meta-learning approach was presented in [12], which utilized MAML (Model-Agnostic Meta Learning) to achieve higher accuracy compared to standard CNN models on a Land Cover classification task. The meta-learning model learns optimal weight initializations to better adapt to tasks without much data. The drawbacks of meta-learning approaches comes from the difficulty of implementation and training, and the in-domain performance of the meta-learning approach tends to be weaker.

However, more traditional deep learning models are have been one of thee most popular approaches to land cover classification. Convolutional models have been perhaps the most common approach given the image-based nature of the problem and have achieved great performance [2] [10], but its robustness on out-of-domain data is less established. Similarly, versions of RNN and LSTM models have been used successfully to classify time-series land cover image data [3] [5]. Transformers are a more recent type of attention [11] based deep learning model that have less commonly been used for land cover classification tasks, but have still seen some applicability, especially in time-series image data [14] [6] where longer dependencies are present. In general, for all these standard deep learning approaches, out-of-domain classification remains a problem. This is what we seek to explore in this paper, to determine which of these approaches lends itself best to out-of-domain generalization on our dataset.

One interesting approach relevant to standard deep learning models is representation learning, which has been used in conjunction to achieve higher accuracies. Tile2Vec [4] is an unsupervised representation learning algorithm that learns continuous representations for individual tiles (small image patches), and has been utilized with a predicting noise as a feature extractor for improved performance on land cover classification [9]. Other self-supervised methods have demonstrated effectiveness at learning feature representations for land cover mapping tasks as well [1]. The results demonstrate the usefulness of representation learning for generalizing tasks, but the lack of high-resolution imagery in our data means that a representation learning approach may not be feasible using our dataset without significant modification.

3. Dataset and Features

3.1. Data

We use the SDG 15: Life on Land datasets from SustainBench [13], which are among several datasets constructed to target various SDGs (sustainable development goals, as stated by the UN). The 2 datasets in this collection each are geared towards a specific kinds of few-shot techniques: one is designed for transfer learning, and the other is designed for representation learning. We intend to use the transfer learning dataset, which is comprised of satellite images from the MODIS satellite collected over the course of 2018, containing 500 points sampled from each of 692 10km×10km regions around the globe. For each point, we have the MODIS Terra Surface Reflectance 8-Day time series for 2018, which contains information for 7 bands, yielding an input dimension of 7×46. Time series coordinates are thus discretized by 8 days per measurement. These 7 bands are Red, Green, Blue, Short-waved Infrared 1, Short-waved Infrared 2, Thermal, and Near-infrared. The labels are the land cover classification for each image.

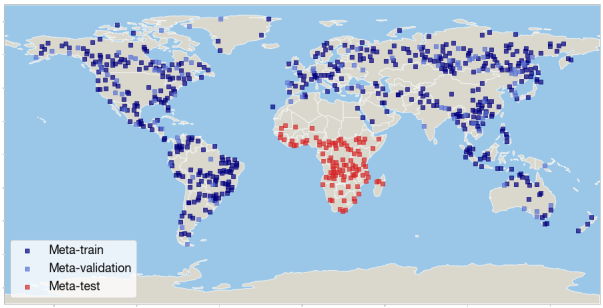

Figure 1. Distribution of meta-train, meta-validation, and meta-test data. We see the meta-test represents out-of-domain data

This dataset has a meta-train, meta-validation, and meta-test split. The meta-test data consists of data taken from Sub-Saharan Africa while the meta-validation data is from approximately the same distribution as the training data. As the naming of the data splits suggest, this dataset was designed for meta-learning, but the data splits also lend themselves well to standard deep learning. In this sense, the meta-train set represents the in-domain training data, the meta-validation represents in-domain validation set, and the meta-test represents an out-of-domain validation set. Comparing the in-domain with the out-of-domain set is important for understanding how well our algorithm is at generalizing to out-of-domain data. In total, there are 210774 training points, 45916 in-domain validation points, and 49721 out-of-domain validation points.

3.2. Pre-Processing

Although the data was provided to us, meaning we didn't have to use any satellite imagery downloading libraries, there are many pre-processing steps we took with the data. For example, many of the images had missing data in certain channels and days in the time series, so we had to clean the data extensively. In addition, we normalized the images, normalizing across the mean and standard deviation of each

Index	Land Cover
0	Evergreen Broadleaf Forests
1	Permanent Wetlands
2	Croplands
3	Permanent Snow and Ice
4	Urban Lands
5	Evergreen Needleleaf Forests
6	Woody Savannas
7	Closed Shrublands
8	Deciduous Needleleaf Forests
9	Grasslands
10	Open Shrublands
11	Savannas
12	Deciduous Broadleaf Forests
13	Cropland Natural Vegetation
14	Barren
15	Mixed Forests
16	Water Bodies

Table 1. Land Cover Classes in Data

of the 7 channels using Eq. (1) for each channel i [??? add info about why we normalized]. For the in-domain and out-of-domain validation datasets, we used the same mean and standard deviation from the training set. We also assigned numeric labels from 0-16 for each of 17 output classes, detailed in Tab. 1

$$\hat{X}_i = \frac{X_i - \mu_i}{\sigma_i} \tag{1}$$

4. Methods

For our experiments, we focus on three deep learning classifiers: a CNN model, an LSTM model, and a Transformer model. We will train each of these models on the in-domain data, and evaluate the performance of each of the models on both the in-domain data and out-of-domain data. As a baseline for comparison, we train a KNN classifier on the in-domain training data. For each model, we test various hyperparameters to optimize training and attempt to achieve the best possible performance for each classifier before comparing their performances relative to the other models.

4.1. K-Nearest Neighbors Baseline

To establish a straightforward baseline, we utilized a KNN model with $K = 3$. KNN classifiers, which classify examples based on the K closest datapoints from the training set, are relatively simple to implement and do not require significant training time, despite having a long test time, while still yielding reasonable results that can serve as a solid base of comparison for deep learning approaches. More specifically, we see that train time is fast because all we have to do is provide the data to the model. Prediction is slow in comparison because this is where the distance comparisons happen between all other points on the predicted inputs. We elected to use $K = 3$ because it yielded the best overall performance for our model. We use standard Euclidean distance for measuring the distance between any two given points in the feature space, as shown in Eq. (2). We implemented this model using [8].

$$\text{Distance}(p, q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \tag{2}$$

4.2. Convolutional Neural Network Approach

Our CNN model consists of the architecture outlined in Fig. 2. These convolution and max-pool layers take place in a 1-dimensional space, given that our data consists of time series of individual pixels (of shape [num_channels $\times$ num_days]) rather than full 2-dimensional images (which would be shape [[num_channels $\times$ height $\times$ width]). We start out with 64 5$\times$1 filters and a padding of 2 for the first convolution, decrease to 32 3$\times$1 filters after the second convolution with 1 padding, and decrease again to 16 3$\times$1 filters after the third convolution with 1 padding. Using 2-padding with filter size of 5 and 1-padding with filter sizes of 3 allow the data to maintain the same number of dimensions within each convolution. Instead, we rely on the maxpooling layer for decreasing the dimensions of the data, having a window of 3, a stride of 1, and 1 padding.

Despite not working directly with full 2-dimensional images, we are still handling pixels with a certain amount of channels over a range of time, and a 1D CNN would still be useful in this situation because it can effectively learn features and patterns within the pixel time series while being computationally efficient. Convolutional models are broadly applicable to image-based data (our pixel time series data are sampled from full images and thus can still be considered image-based), and 1D CNNs are widely used for time-series data. Thus, using a 1D CNN for our image-based pixel time-series data is a natural and logical choice.

4.3. Long Short-Term Memory Approach

Given the sequential nature of our data as the time-series of a pixel over the course of a year, an LSTM-based classifier might be effective way to learn more complex and longer-term relationships within the different components of a single data point.

LSTM models are recurrent neural networks that maintain an additional cell state in addition to the hidden state. This cell state encodes information that we carry over from previous states, and is maintained using four gates that determine how the cell is updated (e.g. what information is

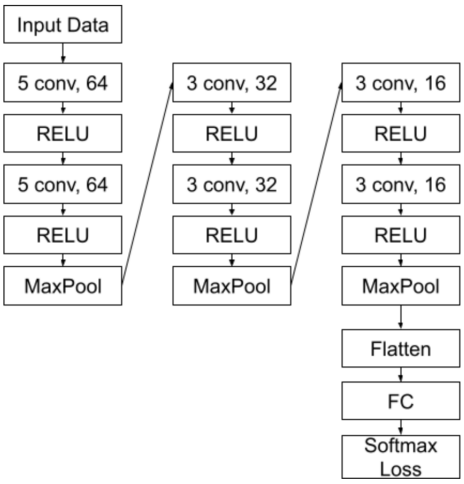

Figure 2. CNN Architecture

kept and what information is erased) and how the cell is utilized in the calculation of the hidden state. An LSTM layer functions as follows: Starting from an initial hidden state and an initial cell state, the hidden state and the input at the current step are are used together with some weights and a bias term using Eq. (3) to compute a vector a of size $4h$ where h is the size of the hidden state. a is then divided into 4 separate vectors, each with an activation function applied as in Eq. (4) to obtain the 4 gates. These 4 gates are then used in Eq. (5) and Eq. (6) to calculate the current cell state and the current hidden state . The process is then repeated until the end of the sequence.

$$a = W_x a + W_h h + b \tag{3}$$

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ tanh \end{pmatrix} a \tag{4}$$

$$c_t = f \odot c_{t-1} + i \odot g \tag{5}$$

$$h_t = o \odot tanh(c_t) \tag{6}$$

In order make a prediction for classification, we flatten the last hidden layer and run it through a fully connected layer to produce a scores for the 17 outputs.

4.4. Transformer Approach

Attention based models have achieved great results in recent years, and despite being designed for NLP tasks, transformer models have been successfully applied to vision problems. Again, since our data contains time-series of pixel data over the course of a year, attention would seem to be a useful method to encode dependencies and learn what parts of the input are relevant to certain classifications.

Our Transformer model consists of 3 encoding layers, each of which is made up of self-attention combined with a feed-forward network. The result of the 3 encoding layers is then passed through a fully connected layer to interpret the results of the self-attention and provide scores for classification.

Self-attention allows the inputs to attend to other parts of the input via a system of keys, queries, and values. Each part of the input (in this case, each pixel state with 7 channels) gets multiplied by a set of key weights W_k, query weights W_q, and value weights W_v. These products are then used to calculate attention scores across all the inputs using Eq. (7).

$$Y = \text{softmax}\left(\frac{(XW_q)(XW_k)^\top}{\sqrt{D}}\right)(XW_v) \tag{7}$$

In order to include more expressivity in the attention calculation, we use multi-headed self-attention instead of general attention. The difference is that multi-headed self-attention performs multiple parallel self-attention calculations on the input, which are then concatenated to form one full attention score. For our model, we use 7 heads (1 head for each channel) since the number of heads needs to divide the input "embedding" dimension exactly.

Within the multi-headed attention step, we also pass each input through a positional encoding layer. This is because attention cannot by default take advantage of sequence of inputs since there are no recurrences or convolutions. As a result, position needs to be encoded through an additional embedding layer. To do this, we define a matrix $P \in \mathbb{R}^{l \times d}$, where $l = 46$ and $d = 7$, and use Eq. (8) to encode the sequential information. When run through the model, we add P to the data.

After the multi-headed attention step, each encoding layer also runs through a feed-forward network. We fine-tuned the dimension of the feed-forward layer, concluding with an optimal value of 100.

$$P(i,j) = \left\{ \begin{array}{ll} \sin\left(i * 10000^{-\frac{j}{d}}\right), & \text{if j is even} \\ \cos\left(i * 10000^{-\frac{j-1}{d}}\right), & \text{otherwise} \end{array} \right\} \tag{8}$$

4.5. Loss Function

Since we are framing this as a multi-class classification problem, we find that using cross-entropy loss, detailed in Eq. (9), is most suitable for the n classes scenario, where $\hat{y}_i$ is the predicted data and y_i is 1 or 0 depending on whether the label is correct. The softmax function in Eq. (10) is applied to the outputs from the model before calculating the loss to condense the results into a probability between 0 and 1. All our deep models are implemented in Pytorch [7]

$$L = -\frac{1}{n}\sum_{i=1}^{n} y_i * \log \hat{y}_i \tag{9}$$

$$\sigma(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{10}$$

5. Experiments and Results

5.1. Evaluation Metrics

To decide our evaluation metrics, we considered which metrics would be most appropriate for a multi-class classification setting. We are using the following metrics:

- *Accuracy*, identified by Eq. (11) measures the amount of correct class predictions over the whole number of predictions.
- *F1-Score* is a harmonic mean of the precision and recall, established by Eq. (12) Eq. (13) Eq. (14). We calculate the F1-score of each class and take the mean over all classes. This metric does not reward True Negative classifications.
- *Average Precision (AP)*, identified by Eq. (15), is the weighted mean of precisions achieved at each threshold n, using increase in previous recall as the weight. We calculate the AP of each class and take the mean over all classes, resulting in the Macro AP (mAP).
- *Macro AUROC* is the area under the Receiver Operating Characteristic Curve, which measures false positive rate against true positive rate. We determine the AUROC for each class and take the mean over all classes.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{11}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{12}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{13}$$

$$\text{F1} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{14}$$

$$AP = \sum_n (\text{Recall}_n - \text{Recall}_{n-1}) * \text{Precision}_n \tag{15}$$

5.2. Model Experiments

For each of the deep models we experimented with, we used a batch size of 64 and trained every model under 10 epochs. For each model, we fine-tuned the model specific hyperparameters (such as architecture, dimensions, etc.) for best performance and report our best results. We experiment later with more general parameters on our best model. The model-specific hyperparameters we ultimately ended up using follow the model architectures discussed in Sec. 4. We also fine-tuned the learning rate of each of the models, with the Transformer model having an optimal learning rate of 0.001, the LSTM model having an optimal learning rate of 0.003, and the Convolution model having an optimal learning rate of 0.002. We also uniformly apply the SGD optimizer with Nesterov momentum for training the models, with Nesterov momentum detailed in Eq. (16) Eq. (17). Although we experimented with several optimizers, we only report SGD with Nesterov momentum since it was the best performing. SGD with Nesterov momentum is similar to the standard SGD update, with the addition of a velocity term v_t to prevent the gradient from getting stuck at local minima and saddle points, while also looking ahead and calculating the gradient where the velocity leads to improve convergence of training.

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t) \tag{16}$$

$$x_{t+1} = x_t + v_{t+1} \tag{17}$$

Looking at the results in Tab. 2, we can see that all 3 of the deep learning models significantly outperformed the baseline KNN model. Within the three deep learning models, the CNN and LSTM classifiers outperformed the Transfomer model. This may be due to the fact that the dimension of a single input into the Transformer is not very large (our 7×46 datapoints translates to an single input dimension of 7 and a sequence length of 46). Thus, there may not be enough information there for the transformer model to properly learn dependencies, especially with the long sequence length. While the in-domain validation accuracy for the Transformer lags behind the CNN and LSTM, the out-of-domain test accuracy is much closer (see Tab. 3). However, this is likely due simply to a favorable distribution of data in the test set where a few classes that the classfier performs well on dominates the data, as the out-of-domain test F1 score is significantly lower for the Transformer model.

The out-of-domain confusion matrices for the best LSTM model (Figs. 5 and 6) and the CNN models are very similar. But, there are some notable differences with the confusion matrices for the Transformer confusion matrix (Fig. 3). The Transformer model often incorrectly classifies data with a true label of 0 ("Evergreen Broadleaf Forests") as label 15 ("Mixed Forests"), as opposed to the CNN and LSTM models which rarely do so. Interestingly, the incorrect labeling from the Transformer model closely reflects the distribution of data in the training set, where "Mixed Forests" about equally as often as "Evergreen Broadleaf Forests". Moreover, the Transformer falsely classifies over 4500 test examples with label 2 ("Croplands"), which is something the LSTM model does fewer than 1200 times.

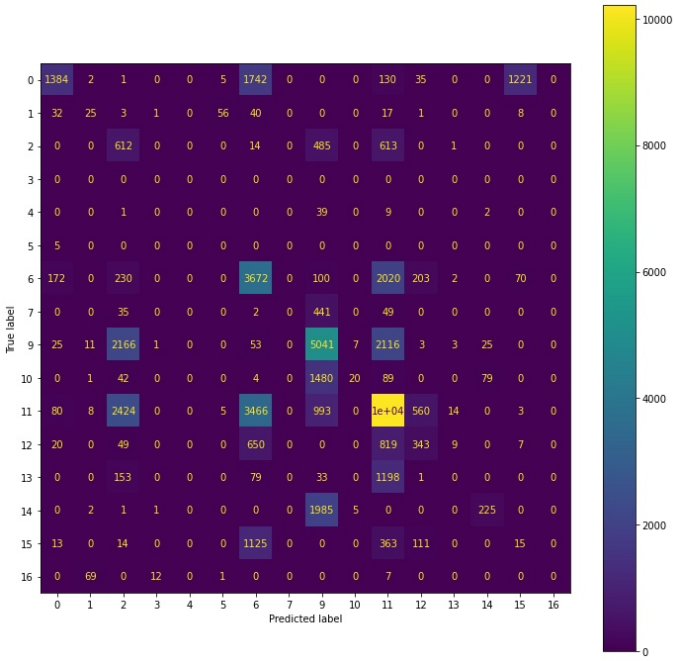

Figure 3. Out-of-Domain Confusion Matrix for Transformer

Model	Accuracy	F1 Score	mAP	AUROC
KNN	0.445	0.404	0.679	0.230
ConvNet	0.654	0.552	0.755	0.378
LSTM	**0.657**	**0.591**	**0.781**	**0.408**
Transformer	0.547	0.376	0.664	0.242

Table 2. Unseen In-Domain Model Results

Model	Accuracy	F1 Score	mAP	AUROC
KNN	0.359	0.278	0.697	0.174
ConvNet	**0.520**	0.423	0.757	0.295
LSTM	0.517	**0.467**	**0.791**	**0.324**
Transformer	0.509	0.275	0.693	0.189

Table 3. Unseen Out-of-Domain Model Results

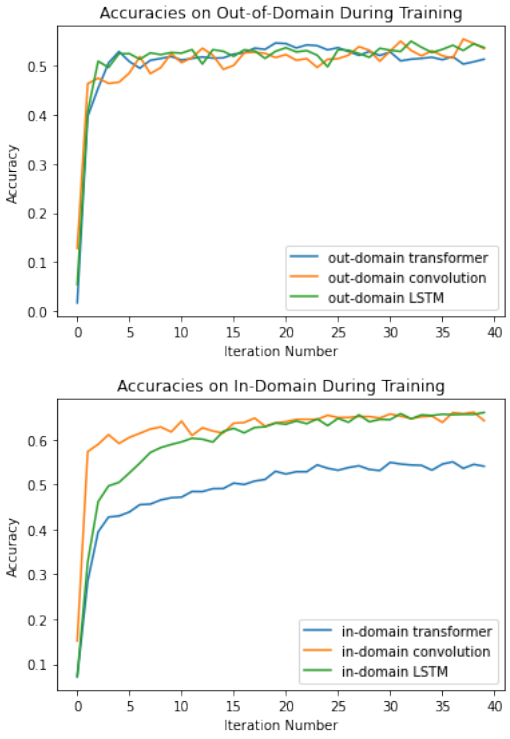

Figure 4. Validation Accuracies by Model

Again, this reflects the distribution of the training data, where "Croplands" appear relatively more frequently. Another example of this includes incorrectly classifying label 10 ("Barren") examples as label 9 ("Grasslands"). This indicates that the Transformer model was less adaptable to changes in the distribution of data between the in-domain training and out-of-domain test sets.

On the other hand, the CNN and LSTM models had very similar performances, with the LSTM classifier slightly outperforming the CNN. Due to the sequential time-series nature of our pixel data, it is likely that the LSTM better learns the longer-term dependencies between the pixel states at different parts of the year (as LSTM models are designed to do), wheres the CNN model with the kernel size of 5 allows for at most a contiguous window of size 5 for which different parts of the input can interact with each other. Overall, we can see in Fig. 4 that the accuracy on the test set quickly flattens out and oscillates around without consistently increasing. This is indicative that additional measures need to be taken in order to increase out-of-domain performance, whether it be utilizing more complex models that adapt better or adding additional regularization strategies.

5.3. Additional Ablation Studies

In addition to running experiments regarding our four primary models, we also conducted ablation studies to precisely evaluate the effect of these general hyperparameters or training choices on both in-domain and out-of-domain performance. We conducted these experiments on the best-performing LSTM model, only modifying one aspect at a time for more precise performance analysis.

5.3.1 Normalization

In Tab. 4 and Tab. 5, we analyze the effects of our choice to normalize the validation data by the training mean and standard deviation. In the TrainNorm case, we maintain our current normalization strategy while in the DatasetNorm

Normalization	Accuracy	F1 Score	mAP	AUROC
TrainNorm	**0.657**	**0.591**	**0.781**	**0.408**
DatasetNorm	0.629	0.564	0.741	0.383

Table 4. Unseen In-Domain Normalization Results

Normalization	Accuracy	F1 Score	mAP	AUROC
TrainNorm	**0.517**	**0.467**	**0.791**	**0.324**
DatasetNorm	0.297	0.253	0.754	0.177

Table 5. Unseen Out-of-Domain Normalization Results

case, we instead normalize the validation sets by their respective mean and standard deviations, rather than that of the training set. We can see that the TrainNorm case results in a higher performance across the board, especially with a significantly higher performance on the out-of-domain set. This makes sense, it is generally best practice to scale all the data by the same factors, and also to avoid touching the validation and test sets.

When we scale/normalize the validation and test set differently from the training sets, we change the characteristics of the data completely and affect the performance of our learned model. Especially since the the out-of-domain test set has a very different distribution of labels compared to the training set, normalizing by the very different statistics of the test set would essentially alter the nature of the test set entirely. The model learns which features and values to care about when running on the normalized training data, and by normalizing the test set using test statistics instead of training statistics, we lose information on certain values which could have been important to accurate classification. This explains why the performance of DatasetNorm on the validation data is lower but much closer to the TrainNorm performance, since the distribution of the in-domain validation data is closer to the distribution of the training data than the test data is.

5.3.2 Batch Size

In Tab. 6 and Tab. 7, we evaluate the effect of batch size on the performance of our LSTM model. We can see that the batch size of 32 generally results in the highest performance across the majority of metrics (except for accuracy in the out-of-domain test set). However, during the process of training these models, we find that the batch size off 32 results in a substantially longer training time, and due to the comparable results obtained using a batch size of 64, we continue experimentation with a batch size of 64 for the sake of efficiency.

Batch Size	Accuracy	F1 Score	mAP	AUROC
32	**0.666**	**0.595**	**0.797**	**0.419**
64	0.657	0.591	0.781	0.408
128	0.639	0.565	0.763	0.385

Table 6. Unseen In-Domain Batch Size Results

Batch Size	Accuracy	F1 Score	mAP	AUROC
32	0.511	**0.473**	**0.809**	**0.332**
64	**0.517**	0.467	0.791	0.324
128	0.529	0.441	0.774	0.316

Table 7. Unseen Out-of-Domain Batch Size Results

p(drop)	Accuracy	F1 Score	mAP	AUROC
0	0.657	0.591	**0.781**	0.408
0.1	**0.666**	**0.594**	0.776	**0.417**
0.4	0.661	0.587	0.779	0.406

Table 8. Unseen In-Domain Dropout Results

p(drop)	Accuracy	F1 Score	mAP	AUROC
0	0.517	0.467	0.791	0.324
0.1	**0.547**	**0.476**	**0.793**	**0.333**
0.4	0.524	0.449	0.780	0.321

Table 9. Unseen Out-of-Domain Dropout Results

5.3.3 Dropout

Finally, in Tab. 8 and Tab. 9, we conduct experiments to see how adding regularization to our model in the form of dropout may improve performance on in-domain and out-of-domain unseen examples. These experiments also give us further insight into the extent in which our model is overfitting. We can see from the results that a dropout value of $p = 0.1$ results in increased performance in both the validation and test sets. The test set performance is notably increased, with accuracy going up by 3 percent and also improving in all other metrics. This indicates some overfitting in our model, and dropout serves as an effective regularizer to reduce the amount of overfitting and increase performance on the unseen validation and test sets. In this sense, regularization through dropout can help prevent overfitting improve performance on out-of-domain unseen data in addition to in-domain unseen data.

5.4. Analysis of Best Overall Model

Overall, we decided our best model that most balanced performance and runtime was the LSTM model with batch

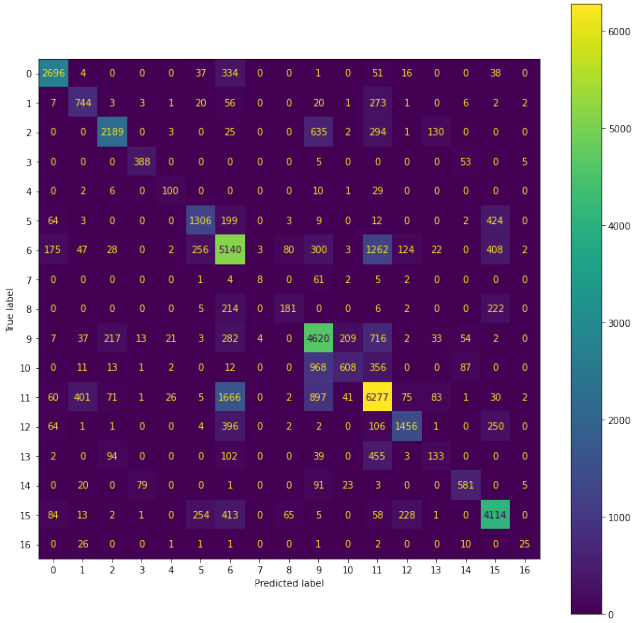

Figure 5. In-Domain Confusion Matrix for Best Performing Model

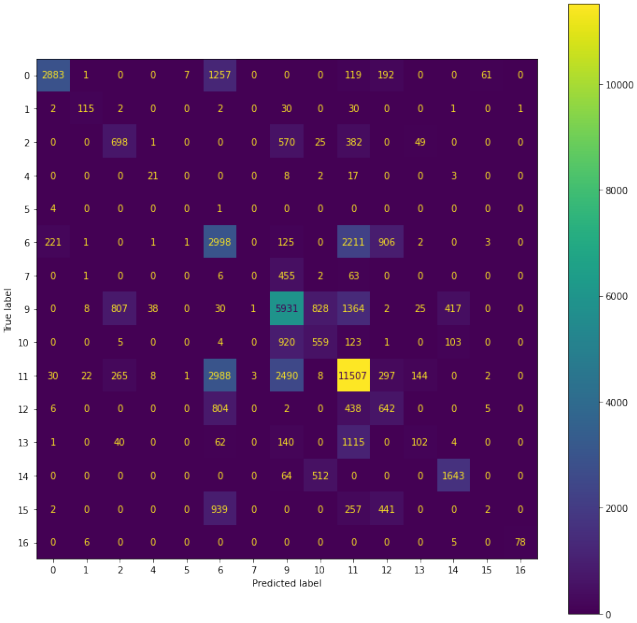

Figure 6. Out-of-Domain Confusion Matrix for Best Performing Model

size 64 and dropout of 0.1. In Fig. 5 and Fig. 6, we have plotted the confusion matrices for this model on the in-domain and out-of-domain data. Many of the common mistakes for our best LSTM classifier makes sense in the context of the labels: Looking at Fig. 6, we can see that some commonly missed examples are label 6: "Woody Savannas" incorrectly labeled as label 11: "Savannas" (and vice versa), and label 9: "Grasslands" being labeled as label 2: "Croplands" (and vice versa). Thinking about the general colors and characteristics associated with each of these land cover types, we can see that many of these mis-labelings can be attributed to outright similarities between the satellite pixels/data points themselves. This is not to say that the LSTM model is not vulnerable to changes in the distribution as we observed the Transformer model to be. Some of the frequent incorrect labels in the confusion matrix (e.g. falsely classifying many test examples as label 6: "Woody Savannas") are reflected by the distribution of the in-domain training data, which contains a much higher frequency of "Woody Savannas" compared to the out-of-domain test set.

6. Conclusion/Future Work

In the end, we find that the LSTM classifier performed the best on the in-domain validation and out-of-domain test sets, slightly outperforming the CNN classifier and significantly outperforming the Transformer model. This is likely because the sequential nature of the time-series pixel data better lends itself to a classifier that can learn longer-term dependencies, like an LSTM model. Despite the ability of Transformers to use attention and learn such dependencies, the size of each input might not be large enough to contain adequate (embedding dimension of 7, sequence length of 46) information for the Transformer model to encode a long sequence of 46 pixel states.

Generally analyzing the results, we see that many of the classification errors reflect the difference in the distribution of the in-domain training set and the out-of-domain test set, where the models often mislabel test examples with a land cover type that is more frequent in the training set than in the test set. Other than that, we can observe some classification errors that are a result of inherent similarity in different types of land cover, in which the models have a tougher time differentiating between land covers with similar colors and characteristics.

Additionally, we see that implementing general deep learning techniques such as normalization using training values, dropout, and batch size adjustment can help improve performance on out-of-domain datasets in addition to in-domain. Nevertheless, if we had more time and compute, there are more experiments we would like to run to potentially improve the performance on the out-of-domain data, such as running deeper and more complex models and also investigating other techniques such as few-shot fine-tuning on out-of-domain data. We would also consider incorporating more significant generalization techniques such as representation learning combined with our existing models.

7. Contributions and Acknowledgements

Both Matt and Jack contributed equally to the completion of this project. Matt: Data Processing, Architecture Design, Experimentation, Paper Jack: Data Processing, Model Fine-tuning, Experimentation, Paper. We would also like to thank our mentor Bohan Wu for his support and guidance throughout this process as well as the CS231N staff for their guidance in this course.

References

[1] Rahul Ghosh, Xiaowei Jia, Chenxi Lin, Zhenong Jin, and Vipin Kumar. Clustering augmented self-supervised learning: Anapplication to land cover mapping, 2021. 2

[2] Yunfeng Hu, Qianli Zhang, Yunzhi Zhang, and Huimin Yan. A deep convolution neural network method for land cover mapping: A case study of qinhuangdao, china. *Remote Sensing*, 10(12), 2018. 2

[3] Dino Ienco, Raffaele Gaetano, Claire Dupaquier, and Pierre Maurel. Land cover classification via multitemporal spatial data by deep recurrent neural networks. *IEEE Geoscience and Remote Sensing Letters*, 14(10):1685–1689, 2017. 2

[4] Neal Jean, Sherrie Wang, Anshul Samar, George Azzari, David Lobell, and Stefano Ermon. Tile2vec: Unsupervised representation learning for spatially distributed data, 2018. 2

[5] Xiaowei Jia, Ankush Khandelwal, Guruprasad Nayak, James Gerber, Kimberly Carlson, Paul West, and Vipin Kumar. Incremental dual-memory lstm in land cover prediction. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, page 867–876, New York, NY, USA, 2017. Association for Computing Machinery. 2

[6] Ioannis Papoutsis, Nikolaos-Ioannis Bountos, Angelos Zavras, Dimitrios Michail, and Christos Tryfonopoulos. Efficient deep learning models for land cover image classification, 2021. 2

[7] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 4

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. 3

[9] Marshal Arijona Sinaga, Fadel Muhammad Ali, and Aniati Murni Arymurthy. Tile2vec with predicting noise for land cover classification. In Teddy Mantoro, Minho Lee, Media Anugerah Ayu, Kok Wai Wong, and Achmad Nizar Hidayanto, editors, *Neural Information Processing*, pages 87–99, Cham, 2021. Springer International Publishing. 2

[10] Andrei Stoian, Vincent Poulain, Jordi Inglada, Victor Poughon, and Dawa Derksen. Land cover maps production with high resolution satellite image time series and convolutional neural networks: Adaptations and limits for operational systems. *Remote Sensing*, 11:1986, 08 2019. 2

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. 2

[12] Sherrie Wang, Marc Rußwurm, Marco Körner, and David B. Lobell. Meta-learning for few-shot time series classification. In *IGARSS 2020 - 2020 IEEE International Geoscience and Remote Sensing Symposium*, pages 7041–7044, 2020. 2

[13] Christopher Yeh, Chenlin Meng, Sherrie Wang, Anne Driscoll, Erik Rozi, Patrick Liu, Jihyeon Lee, Marshall Burke, David Lobell, and Stefano Ermon. Sustainbench: Benchmarks for monitoring the sustainable development goals with machine learning. In *Thirty-fifth Conference on Neural Information Processing Systems, Datasets and Benchmarks Track (Round 2)*, 12 2021. 2

[14] Yuan Yuan and Lei Lin. Self-supervised pretraining of transformers for satellite image time series classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 14:474–487, 2021. 2