Horoma - Block 2

Ishaan Kumar
Department of Computer Science
Université de Montréal, Montreal, QC
ishaan.kumar@umontreal.ca

Alejandra Jimenez Nieto
Department of Computer Science
Université de Montréal, Montreal, QC
alejandra.jimenez.nieto@umontreal.ca

David Abraham
Department of Computer Science
Université de Montréal, Montreal, QC
david.abraham@umontreal.ca

Abstract

In this project, we explore deep learning techniques for labelling aerial forest images. This is formulated as a semi-supervised problem where unsupervised techniques are used for learning clustering of tree species and supervision is used to label the clusters.

1 Introduction

We have a collection of aerial forest images and we want to label them with their respective species. Data labelling by human interpreters is costly and time consuming. The goal of this project is to develop a machine learning program that labels images with limited amount of data.

Block 1 projects aimed at constructing a classifier that utilized limited labelled data to train a supervised classifier for this purpose. In this block, the aim is to leverage unlabelled data and modern deep learning techniques to improve performance.

2 Data description

Two datasets have been provided - Training and Validation. The training dataset contains 150700 image samples only while the validation data set contains 480 images and labels. Each image is an 32x32 RGB image with each channel value between [0, 255]. The label for the validation set is a two character label and denotes the tree specie.

These images have been constructed by breaking down a very large image into patches of 32x32. Variant of the datasets with overlapping patches has also been provided. The overalapped training dataset contains 544749 images, whereas the overlapped validation dataset contains 1331 images.

As is seen in figure 1, the class labels in validation dataset are not balanced. This makes our task even more challenging as in the absence of true labels for a particular class, it's very hard to generalize. If we are not careful, our algorithm may even end up learning that each sample belongs to four of the classes for which we have the majority of labels.

Using the overlapped dataset is not the solution here as classes with less labels don't have much imagery to construct overlapped samples out of. Thus, the overlapped dataset has even more disparity among label frequencies and using it may aggravate the problem.

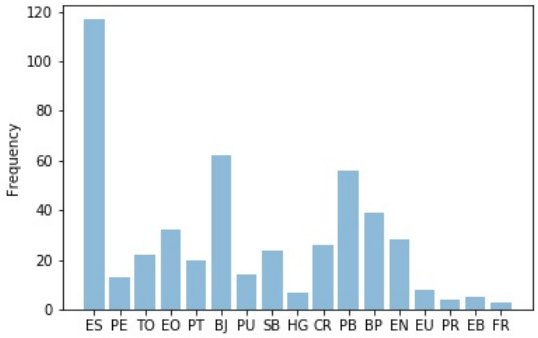

Figure 1: Class frequencies in validation dataset

3 Methods used

Using unlabelled data, all methods first learn a latent representation of the data in a completely unsupervised manner. This step reduces the dimension of our data. Using the learnt embeddings, clusters are learned. To map the learned cluster labels to species labels, validation dataset samples are mapped onto these clusters and a majority vote is taken. This means that some clusters may end up having no mapping at all. Dimension reduction and clustering algorithms used are mentioned below.

3.1 Autoencoder

An autoencoder is a type of neural network that can learn efficient embeddings of data. It usually has two components - encoder and decoder. The encoder takes the original data as input and reduces it to lower dimension representation. The decoder takes the lower dimension representation and tries to recreate the original input. Following variants of autoencoders were tried.

Multilayer In this variant of autoencoder, the encoder and decoder networks are many layers deep, but the number of encoders and decoders still remain one. These networks are good for learning latent representation of the input and are trained to group similar inputs together. Similarity is defined by a distance metric.

Multilayer Convolution These networks are similar to Multilayer autoencoders, the only difference being that instead of fully connected layers in encoder and decoder we use convolution layers. This makes sense for our case as we have image data and all the advantages of convolution layers can be leveraged to learn a better representation.

Variational Instead of correctly guessing the distribution followed by latent vectors, we force the network to fit a prior distribution. This proves to be useful when we want to use the decoder as a generative model. In our case, the knowledge of prior distribution can be exploited to select an appropraite cluster method. For example, if we fix the prior to be a normal distribution, we can use Gaussian Mixture Model algorithm to learn clusters.

Stacked Autoencoders can be stacked to form a deep network by feeding the latent representation of the autoencoder found of previous layer as input to the current layer. The unsupervised training of such an architecture is done one layer at a time. Each layer is trained as an autoencoder by minimizing the error in reconstructing its input. In this variant, each autoencoder layer is a denoising autoencoder. In a denoising autoencoder, some noise is added to the input of the encoder part and than the decoder part tries to reconstruct the real data (data without the noise). This helps the hidden layer to discover more robust features and prevent it from simply learning the identity.

3.2 CNN

CNN learn meaningful representation of image data which can be used for a variety of task. Instead of training a CNN from scratch, transfer learning can be used to fine-tune a pre-trained CNN network

for our task. We use SqueezeNet [3] that is trained on more than a million images from the ImageNet database. The network is 18 layers deep and can classify images into 1000 classes [1].

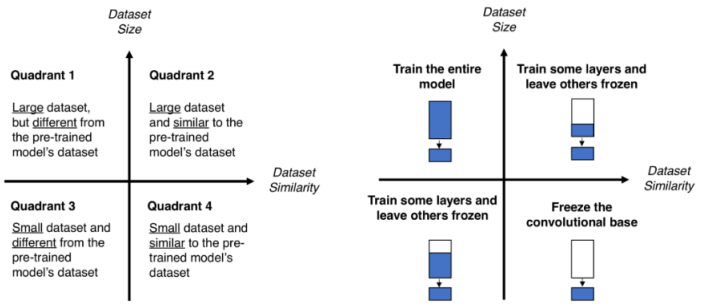

Figure 2: Fine tuning strategies

Figure 2 describes the various fine tuning strategies. As our dataset is large and different from ImageNet, we fall into the Quadrant 1 and train the entire model.

3.3 Kmeans

K-means clustering is a type of unsupervised learning. The goal of this algorithm is to find K clusters in the data. The algorithm works iteratively to assign each data point to one of the K clusters based on provided features. Data points are clustered based on feature similarity. The result of the K-means algorithm are the centroids of the K clusters which can than be used to label new data.

3.4 Gaussian Mixture Model

Gaussian mixture model (GMM), a type of unsupervised clustering, is a probabilistic model that assumes all the data points are generated from a mixture of a finite number K of Gaussian distributions with unknown parameters. With this approach we describe each cluster by its centroid (mean), covariance, and the size of the cluster (weight). One advantage of GMMs over K-means is that this technique is better to deal with clusters that aren't circle or sphere shaped.

4 Algorithm description

4.1 Naive Approach for Autoencoders

In the approach the autoencoder is trained end-to-end, without any layer pretraining. Learnt training data embeddings are used to learn clusters. The clustering algorithm randomly assigns each cluster a label. To classify unseen examples we need a mapping that maps the learned labels to the true labels.

Validation dataset is used to learn this mapping. Validation data embeddings are classified using learned clusters and the true label which has the highest frequency for the random label is assigned to the random label.

4.2 Simultaneous clustering & dim. reduction

Algorithm 1 explains the process in detail. Based on algorithm suggested by [5]Yang et al., the autoencoder is guided by cluster loss to make cluster friendly latent representation.

4.3 Transfer learning with CNN

Algorithm 2 explains the transfer learning algorithm in detail. Our algorithm is a direct implementation of the algorithm proposed by [1]Caron et. al.

[1] https://www.mathworks.com/help/deeplearning/ref/squeezenet.html

Data: Training Data, Validation data
Initialize the auto encoder;
Initialize clusters with training data embedding extracted using initialized auto encoder;
while *Validation loss is decreasing* **do**
  for *each mini-batch of training data* **do**
    Do a forward pass;
    Compute loss using function *compute_loss(data, recon_data, data embedding)*;
    Update weights;
  end
  Update the clusters using the new training data embeddings ;
end
Learn mapping of fake labels to true labels using validation data ;
Function *get_cluster_loss(embedding):*
  Get the closest cluster center point for the embedding ;
  loss = L_2(*embedding - closest cluster center*) ;
end
Function *compute_loss(x, recon_x, embedding):*
  embedding_loss = get_embedding_loss(x, recon_x);
  cluster_loss = get_cluster_loss(x, embedding);
  return embedding_loss + cluster_loss;
end

Algorithm 1: Simultaneous clustering & dim. reduction

Data: Training Data, Validation data
Initialize Squeezenet model;
Optimizer SGD definition with model parameters;
while *Validation loss is decreasing* **do**
  Set model to training mode;
  for *each mini-batch of training data* **do**
    Do a forward pass;
    Learn/Update clusters on CNN embedding ;
    Treat cluster labels as true labels and compute loss ;
    Update weights;
  end
end

Algorithm 2: Clustering assignment as pseudo-labels & CNN Training

5 Experiment Methodology

5.1 Data Treatment

Splitting For training and validation purpose, we use non overlapped data. Each of these datasets is further divided into two datasets. 95% of unlabelled data is used for training embedding and 5% is used as unlabelled validation set to evaluate performance of embedding layer. 50% of labelled data is used for learning cluster label mapping mentioned above and the other half is used as labelled validation set. Labelled dataset is split using stratified sampling.

Augmentation As we have less and unbalanced validation dataset, data augmentation can be used to increase data samples. We have used *RandomRotation*, *RandomHorizontalFlip* and *ColorJitter* torchvision transformation techniques to augment data.

Normalization Each pixel value is normalized between [0, 1].

Class Balancing This technique is used for reducing bias in our classifier. As classes with more labels can sway the decision in a majority vote, it is necessary to balance the classes. One can also use weighted voting to counteract this effect. We upsample data to balance the classes.

5.2 Experiment Details

- All the methods were first trained with naive algorithm and then with simultaneous clustering algorithm and were repeated with toggling data augmentation and class balancing.
- After evaluating performance of various techniques, latent dimension hyper-parameter was tweaked for the best performing technique to observe its effect on performance. Refer *Results* section for more details.
- Number of clusters for the clustering algorithms is always set to the number of unique classes in labelled dataset *i.e* 17. Accuracy and f1 scores can be increased substantially by forcing the algorithm to learn more number of clusters, but that is not a good practice and may lead to over-fitting.
- For all the experiments we use *Adam* optimizer with initial lr=10^{-4}.

6 Results

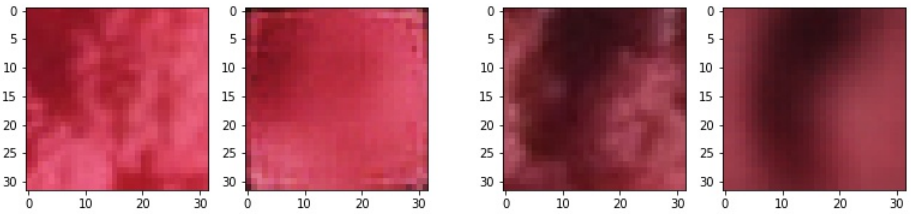

Figure 3: Reconstruction example of (a) CAE with data augmentation (b) CAE with raw data

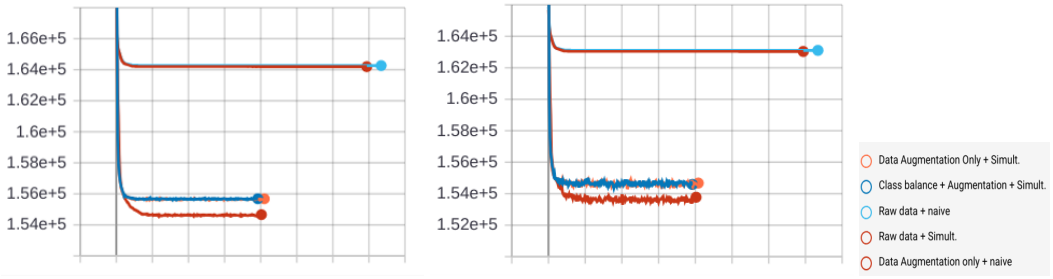

Figure 4: CAE + Kmeans (a) Loss on unlabelled training (b) Loss on unlabelled validation till 367 epochs

6.1 Naive Algorithm

6.1.1 Autoencoders

With the naive approach, the autoencoder is trained using only the training (unlabelled) dataset in an unsupervised manner. Table 1 shows the performance metrics computed on the validation part of the labelled dataset. The best f1 score of 0.341 for this dataset is obtained using a convolutional autoencoder as the embedding model, k-means as the clustering model and without class balancing and data augmentation.

Table 1 shows the performance metrics computed on the validation part of the labelled dataset. The best f1 score of 0.341 for this dataset is obtained using the same model as above.

Embedding Model	Cluster	Class balance	Data augmentation	f1	Acc [%]	ARI	NMI
AutoEncoder	kmeans	No	Yes	0.207	28.7	0.066	0.228
AutoEncoder	kmeans	No	No	0.266	35.4	0.138	0.323
Convolutional AE	kmeans	No	Yes	0.187	27.6	0.08	0.186
Convolutional AE	kmeans	No	No	0.341	41.6	0.186	0.359
Variational AE	kmeans	No	Yes	0.169	24.3	0.035	0.15
Variational AE	kmeans	No	No	0.204	27.6	0.065	0.236
Stacked CAE	kmeans	Yes	Yes	0.125	16.9	0.08	0.212
Stacked CAE	kmeans	No	Yes	0.231	30.5	0.102	0.227
Stacked CAE	kmeans	No	No	0.242	32.9	0.124	0.253
AutoEncoder	gmm	No	Yes	0.16	26.3	0.033	0.2
AutoEncoder	gmm	No	No	0.3	39.9	0.155	0.357
Convolutional AE	gmm	No	Yes	0.185	28.7	0.084	0.205
Convolutional AE	gmm	No	No	0.306	36.6	0.154	0.345
Variational AE	gmm	No	Yes	0.187	28.0	0.07	0.188
Variational AE	gmm	No	No	0.247	32.5	0.108	0.252
Stacked CAE	gmm	Yes	Yes	0.115	15.2	0.058	0.194
Stacked CAE	gmm	No	Yes	0.199	29.5	0.086	0.231
Stacked CAE	gmm	No	No	0.187	29.2	0.093	0.199

Table 1: Performance of naive algorithm on validation part of validation (labeled) dataset.

Embedding Model	Cluster	Class balance	Data augmentation	f1	Acc [%]	ARI	NMI
AutoEncoder	kmeans	Yes	Yes	0.146	20.1	0.094	0.25
AutoEncoder	kmeans	No	Yes	0.224	31.3	0.111	0.242
AutoEncoder	kmeans	No	No	0.271	37.9	0.143	0.321
Convolutional AE	kmeans	Yes	Yes	0.12	16.8	0.064	0.221
Convolutional AE	kmeans	No	Yes	0.192	27.2	0.05	0.186
Convolutional AE	kmeans	No	No	0.335	39.5	0.157	0.342
Variational AE	kmeans	Yes	Yes	0.168	24.7	0.124	0.307
Variational AE	kmeans	No	Yes	0.271	36.1	0.132	0.284
Variational AE	kmeans	No	No	0.294	38.7	0.15	0.326
AutoEncoder	gmm	Yes	Yes	0.146	20.2	0.097	0.243
AutoEncoder	gmm	No	Yes	0.269	36.1	0.116	0.252
AutoEncoder	gmm	No	No	0.282	37.0	0.136	0.316
Convolutional AE	gmm	Yes	Yes	0.096	15.5	0.056	0.189
Convolutional AE	gmm	No	Yes	0.193	27.6	0.046	0.207
Convolutional AE	gmm	No	No	0.26	35.0	0.137	0.312
Variational AE	gmm	Yes	Yes	0.121	18.2	0.104	0.253
Variational AE	gmm	No	Yes	0.234	34.5	0.109	0.241
Variational AE	gmm	No	No	0.285	37.0	0.151	0.322

Table 2: Performance of simultaneous clustering & dim. reduction algorithm on the validation part of the validation (labeled) dataset.

6.2 Simultaneous clustering & dim. reduction

With the simultaneous clustering & dim. reduction approach, the training of the autoencoder is guided by the cluster loss. Table 2 shows the performance metrics computed on the validation part of the labeled dataset. The best f1 score of 0.335 for this dataset is obtained using a convolutional autoencoder as the embedding model, k-means as the clustering model and without class balancing and data augmentation.

6.3 Transfer learning on CNN

Embedding Model	Cluster	Data augmentation	Acc (%)
SqueezeNet & pseudo-labels	kmeans	Yes	18.95
SqueezeNet & pseudo-labels	kmeans	No	13.75
SqueezeNet & pseudo-labels	mini-batch-kmeans	Yes	12.91
SqueezeNet & pseudo-labels	mini-batch-kmeans	No	12.91

Table 3: Best accuracy in validation set for the SqueezeNet CNN using kmeans and mini-batch-kmeans and with and without data augmentation.

6.4 Latent dimension tuning

Table 4 shows the performance for different sizes of latent dimension size of the embedding model for our best performing architecture, which is a convolutional autoencoder with k-means without class balancing and data augmentation. We can not observe any direct effect of the latent dimension size on the performance of the model. As we increase the size, the f1 score and accuracy stays very similar.

Latent dim. size	f1	Acc [%]	ARI	NMI
8	0.298	42.0	0.172	0.372
16	0.287	37.9	0.159	0.335
32	0.341	41.6	0.186	0.359
64	0.295	38.7	0.162	0.334
128	0.321	42.0	0.206	0.383

Table 4: Performance of best model for different latent dimension size

6.5 Comparison with previous blocks

The baseline model of block 1 had an accuracy of 36.91% and an f1 score of 0.33 on the validation set. Our best model has an accuracy of 41.6% and an f1 score of 0.341 on the validation part of the labelled set. We can see that using deep learning to train an embedding model instead of simply using PCA or a similar technique to reduce the dimensionality gives slightly better performance.

6.6 Effective clusters learned

Due to unbalanced classes and majority vote algorithm, the learned mapping from cluster labels to true labels merge some clusters. We observe that despite class balancing and data augmentation, the number of effective clusters remain between 7-10. This is shown in figures 5, 6, and 7. The left panel shows the t-SNE projection of points and kmeans clusters and right panel shows the effective clusters. As is seen in confusion matrix(Figure 8), the best model effectively learn 7 clusters.

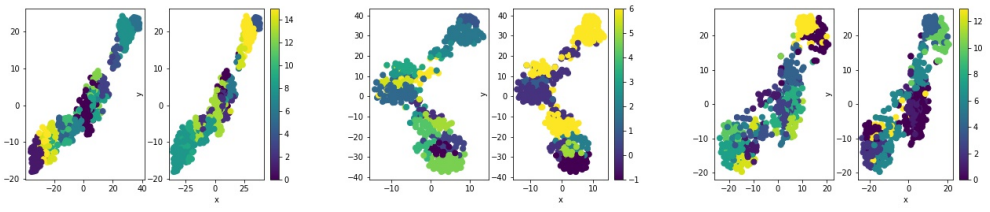

Figure 5: Clusters learned with simultaneous clustering (a) CAE + Kmeans with class balancing and data augmentation (b) CAE + Kmeans with data augmentation, (c) CAE + Kmeans with raw data.

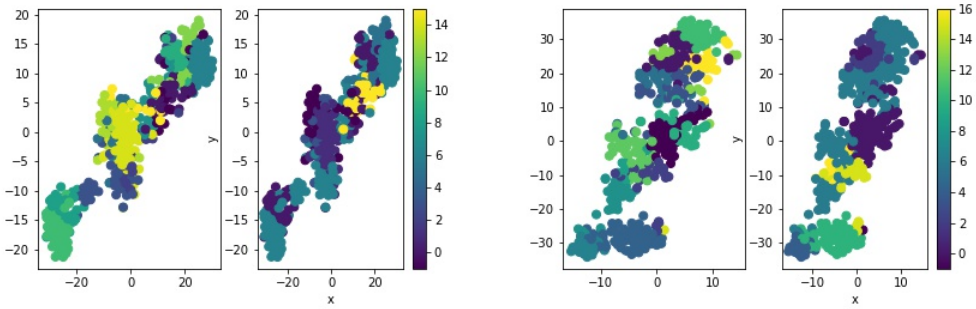

Figure 6: Clusters learned with naive algorithm. (a) CAE + Kmeans with data augmentation, (b) CAE + Kmeans with raw data.

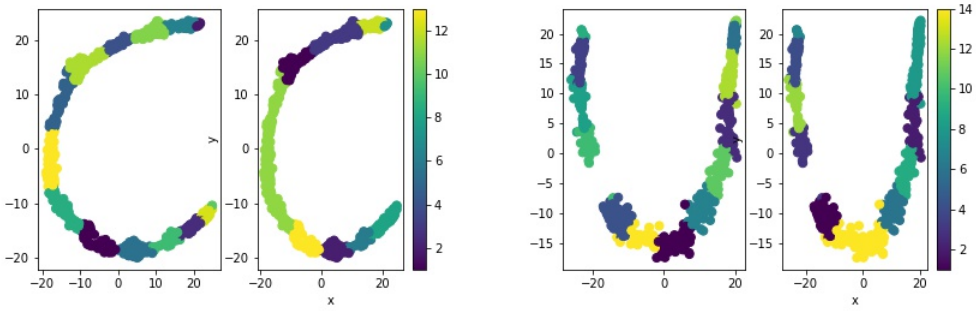

Figure 7: Left panel shows true clusters learned and the right panel shows effective clusters learned using simultaneous clustering algorithm. (a) AE + Kmeans (b) VAE + Kmeans. Both were trained with data augmentation and balanced classes.

6.7 Discussion

Here are some observations on the results:

- We see that Kmeans performs better than Gaussian Mixture with our embedding models. This makes sense for simultaneous clustering algorithm as [5]Yang et al. formulated cluster loss for Kmeans and not Gaussian Mixture Model.
- As data augmentation and class balancing are regularization techniques, they bring down the f1 score and accuracy but prevent score inflation.
- We observe that for AE and VAE naive algorithm performs worse than simultaneous clustering & dim. reduction, but for CAE it's the other way round.
- In our experiments, the latent dimension size of the embedding model has no direct measurable effect on the performance of the model.
- The above two observation lead us to believe that cluster label mapping is the real performance bottleneck. Improvements for learning the correct mapping would lead to better performance.

7 Next steps

- Any method leads to better label mapping would lead to better performance. Boosting would also lead to better performance for classes with less data.
- Super-resolution imaging technique as proposed by [4]Sønderby et al. could be used as a pre-processing step. With reduced noise, the dimension reduction techniques should perform better.

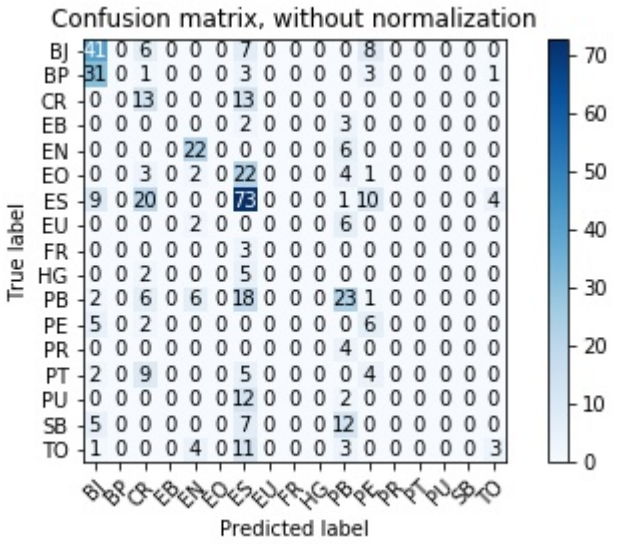

Figure 8: Confusion matrix for our best model

- Perhaps implementing deep clustering with VAE and Gaussian mixtures as suggested by [2]Dilokthanakul et al. could improve performance with GMM.

8 Conclusion

In summary, we tried multiple types of embedding models: a simple multilayer autoencoder, a convolutional multilayer autoencoder, a convolutional variational autoencoder and a stacked denoising autoencoder. All of these embedding models were used with kmeans and GMM as the clustering algorithm. For each of the pair of models we tried training them with a naive approach (i.e. training the embedding model in a completely unsupervised manner) and using a simultaneous learning of the embedding model and the clustering model. We also tried to use a CNN (SqueezeNet) to extract features for the clustering algorithms. All of the methods described were used with and without class balancing and data augmentation.

Our best performing model is the convolutional multilayer autoencoder with kmeans using the naive training approach without class balancing and data augmentation which got an f1 score of 0.341 and an accuracy of 41.6% on the validation part of the labeled dataset.

References

[1] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. *CoRR*, abs/1807.05520, 2018.

[2] Nat Dilokthanakul, Pedro A. M. Mediano, Marta Garnelo, Matthew C. H. Lee, Hugh Salimbeni, Kai Arulkumaran, and Murray Shanahan. Deep unsupervised clustering with gaussian mixture variational autoencoders. *CoRR*, abs/1611.02648, 2016.

[3] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016.

[4] Casper Kaae Sønderby, Jose Caballero, Lucas Theis, Wenzhe Shi, and Ferenc Huszár. Amortised MAP inference for image super-resolution. *CoRR*, abs/1610.04490, 2016.

[5] Bo Yang, Xiao Fu, Nicholas D. Sidiropoulos, and Mingyi Hong. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. *CoRR*, abs/1610.04794, 2016.