Photometric Redshift Estimation

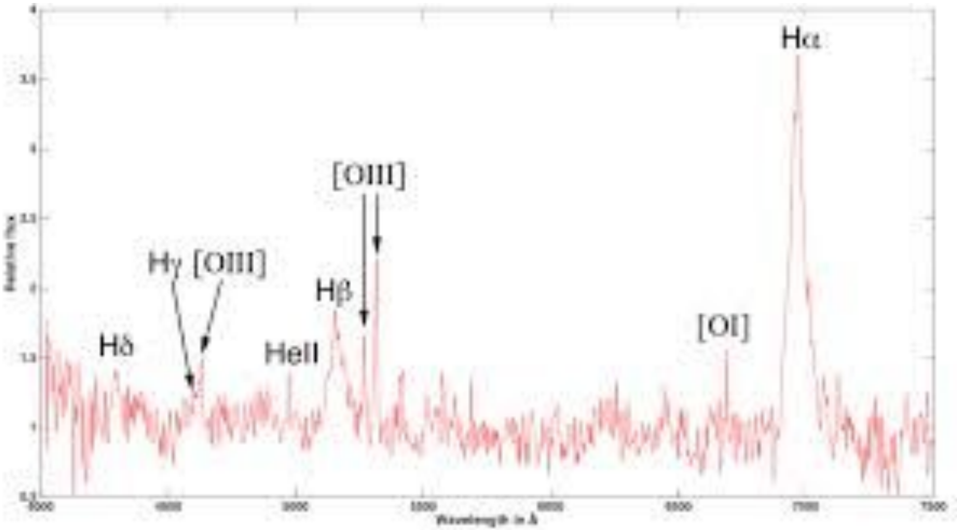
Andrew Engel, Patrick Aleo, Lina Florez, Tsung-Han Yeh
ASTR 596, Prof. Gautham Narayan

Brief Review of Photometric Redshifts

(like seriously ~ 2 min)

Measuring Redshifts Through Spectroscopy is Slow

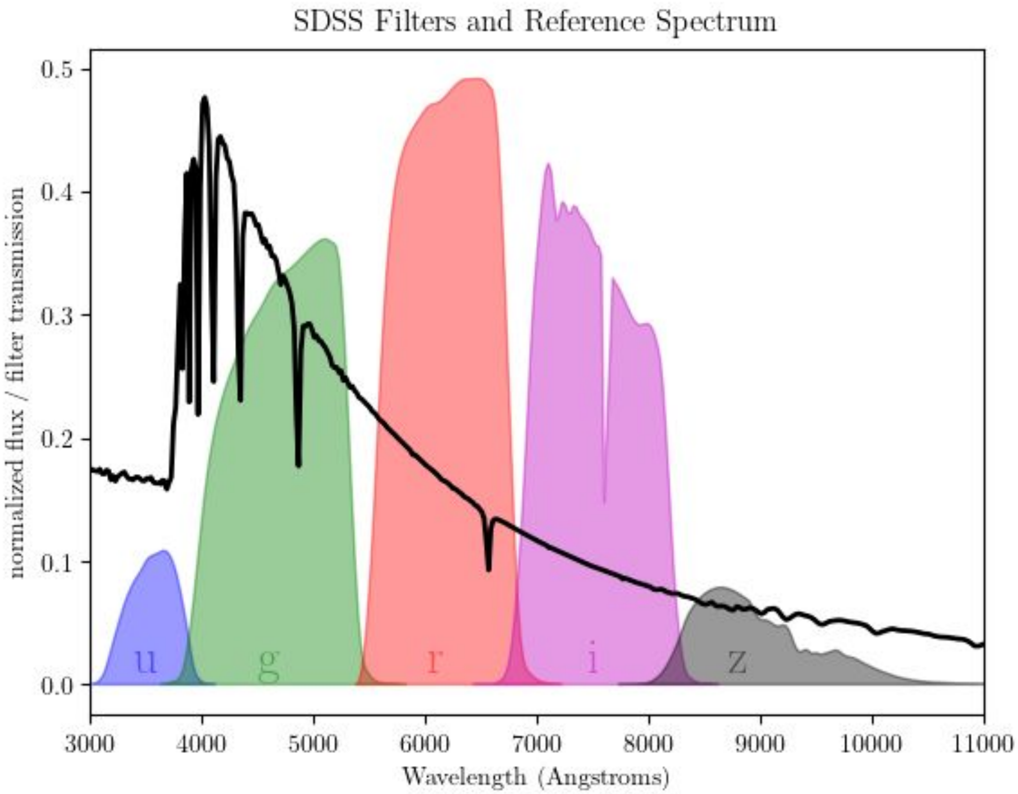
Spectra show shifts in emission/absorption lines, allowing measurement to high accuracy of Z



However Spectrographs are slow in comparison to taking photometric data:

SDSS OG Spectrograph: 640 objects in 45 min. In Comparison: Photometry takes 1 field (left) per ~ 2 minutes

Photometric redshift algorithms estimate Z without spectra



Empirical Redshift Algorithms attempt to use Photometry (images and their derived data products) to try to find a mapping from some feature space to Z space.

However, Large Redshift uncertainties contribute to uncertainties in measurements of cosmological parameters such as in Baryonic Acoustic Oscillations methods. [https://arxiv.org/pdf/1610.09688.pdf]

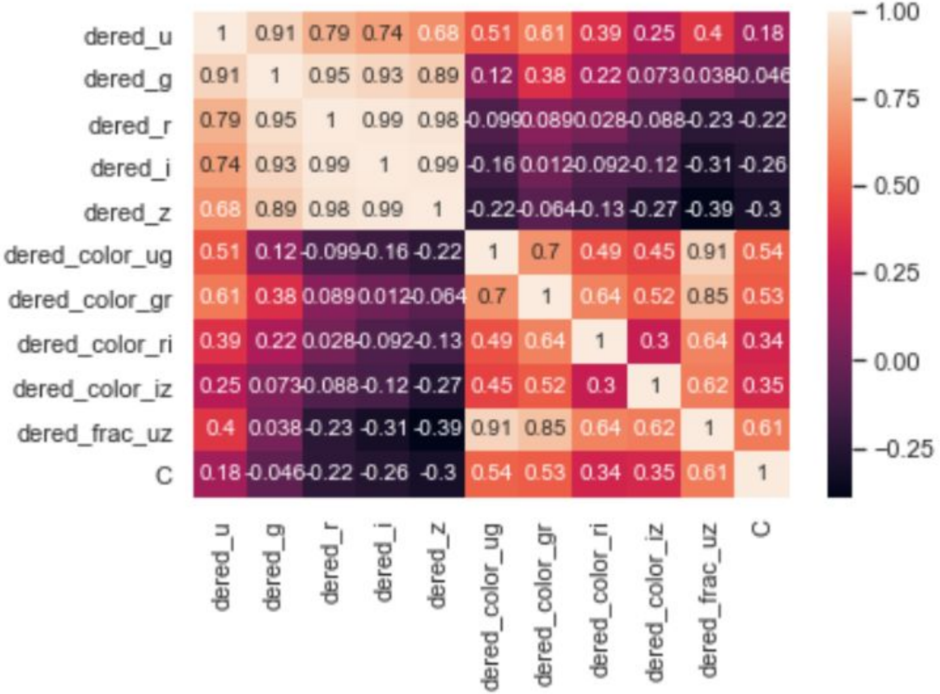
Photometric Redshifts pose a boon by increasing number of objects with known redshift if photo-z errors can be reduced to sub-percent.

Data

SDSS DR12 Galaxies with i-band magnitudes < 25; no mergers, 0<Z<0.4

We queried photometry including u,g,r,i,z band dereddened magnitudes, petrosian radii, PSF fluxes

Each algorithm uses the same train (N=175,000), validation (N=125,000), and test sets (N~250,000)



Metrics

```
#summary stats
residuals = (x-y)/(1+x)
```

Median Absolute Deviation

```
MAD = 1.4826*np.median(abs(residuals - np.median(residuals)))
```

Bias metrics

```
bias = np.mean(residuals)
```

Outlier metrics

```
eta = len(residuals[residuals > 5*MAD])/len(residuals)
```

PIT answers: How well behaved are the output PDFs? (when available)

More than one line, see lecture for example

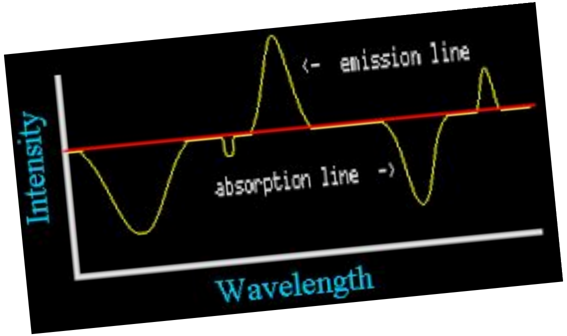
Part 2: ML Photo-Z Algorithms

Supervised K Nearest Neighbors

Random Forest Regression

ANN: Multi Layer Perceptron

ANN: Spectral Encoder



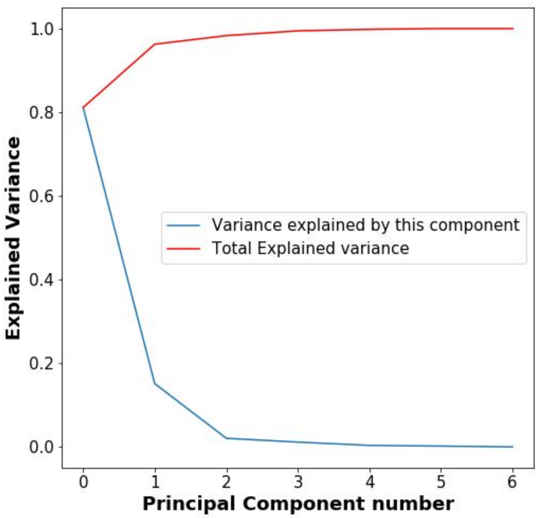
KNN

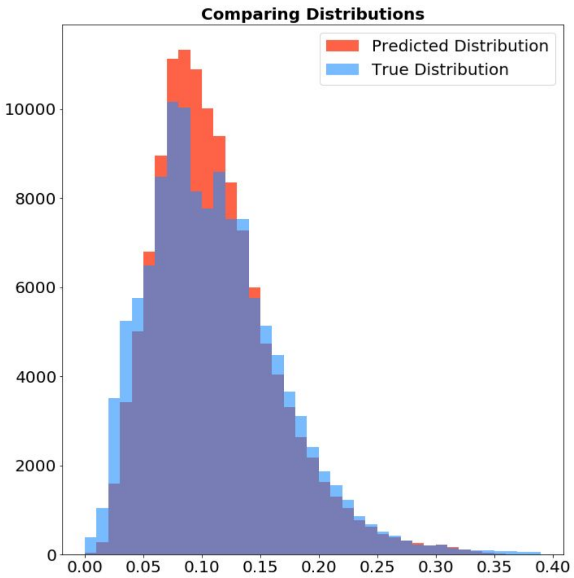
Principal Component Analysis

```
def compute_PCA(input_data, n_components):

    # we're setting the mean as the first component
    mean = input_data.mean(0)
    pca = PCA(n_components)

    pca.fit(input_data)

    # and the explained variance is:
    evals = pca.explained_variance_ratio_
    evals_cs = evals.cumsum()

    #Checking to see if dimensionality was reduced
    pca_transformed = pca.transform(dat)

    return pca_transformed, evals, evals_cs, pca.components_, pca.mean_
```

K-Nearest Neighbors

```
def K_Means_Regressor(og_data, number_of_pc, n_neighbors, test_rows, actualdata_rows):
    #Computing PCA
    pca_data, e, e_cs, components, pca_mean = compute_PCA(og_data, number_of_pc)

    #K-Means Nearest Neighbors
    neigh = KNeighborsRegressor(n_neighbors)
    neigh.fit(pca_data[0:test_rows], df_ztrim.z.iloc[0:test_rows])
    predicted_redshift = neigh.predict(pca_data[test_rows:actualdata_rows])

    return predicted_redshift
```





Data Reduced:

Original data: (589058, 11)

PCA transformed data: (589058, 7)

Number of Principal Components: 7
Number of nearest neighbors: 9
MAD: 0.0168
bias: -0.0006
outlier_fraction_eta pasquet: 0.0017
RMS: 0.0215
outlier fraction LSST: 0.0126

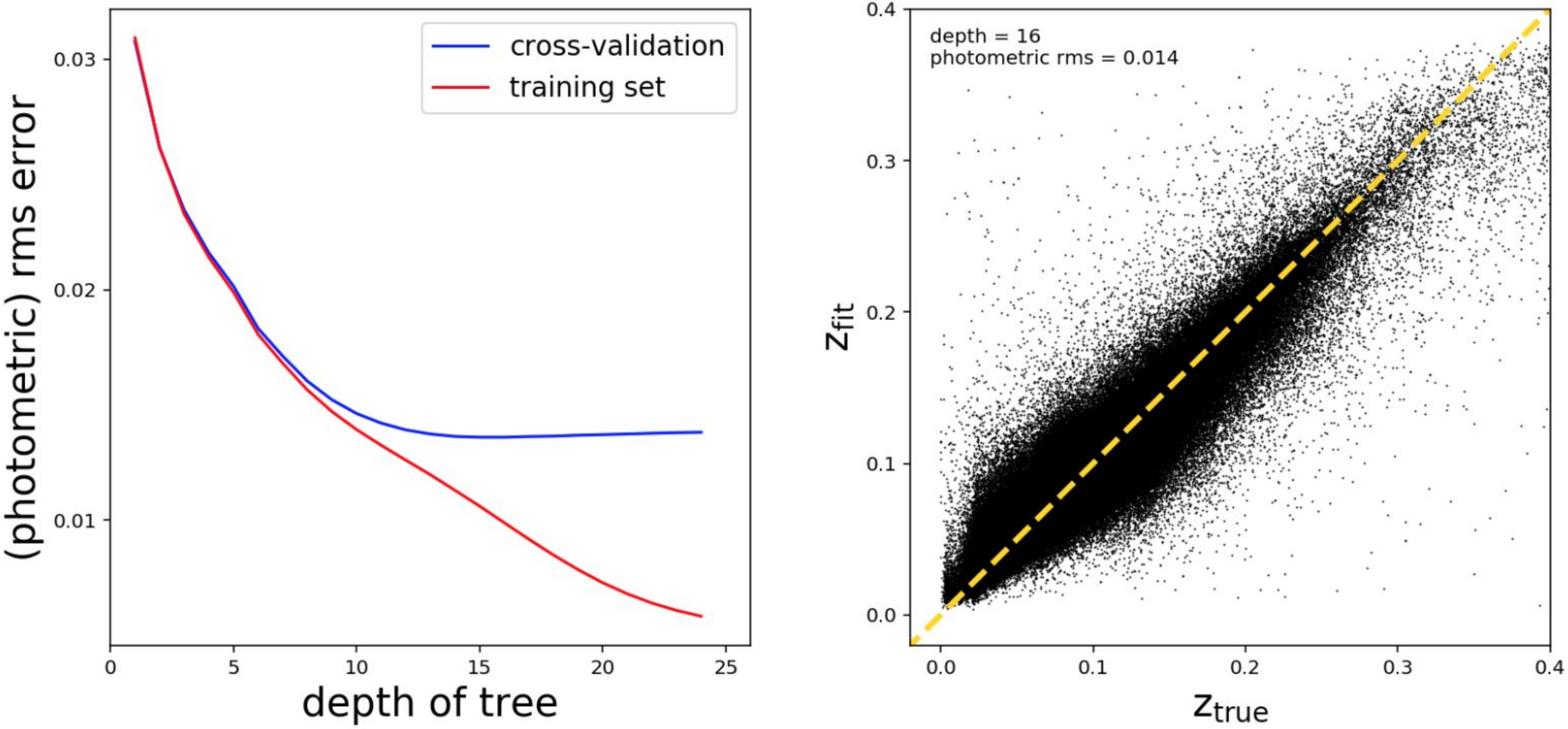
	dered_u	dered_g	dered_r	dered_i	dered_z	dered_color_ug	dered_color_gr	dered_color_ri	dered_color_iz	dered_frac_uz	C
specObjID											
345721668045400064	20.07695	18.21239	17.23913	16.82238	16.49874	1.864559	0.973261	0.416756	0.323633	1.216878	2.958234
345689782208194560	18.76690	17.66344	17.22466	16.92888	16.84306	1.103458	0.438782	0.295778	0.085827	1.114222	1.990836
529188106864191488	19.78869	18.18713	17.25847	16.89089	16.60906	1.601561	0.928652	0.367588	0.281828	1.191439	2.673254
531462996858267648	18.42462	16.54012	15.65375	15.23006	14.89166	1.884495	0.886367	0.423692	0.338399	1.237244	3.322546
535942963008661504	19.67483	18.07697	17.21868	16.78680	16.42058	1.597860	0.858295	0.431875	0.366219	1.198181	2.325654

11 features to predict z

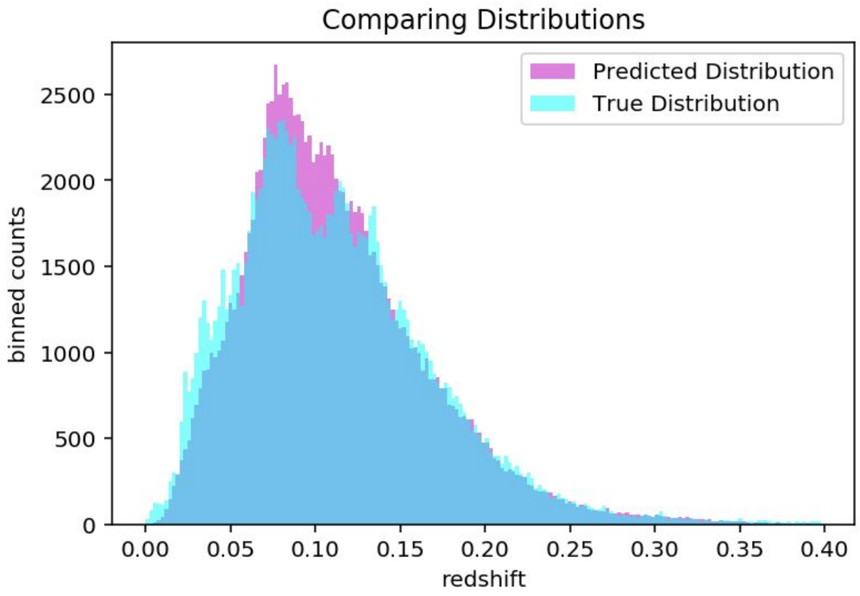
Random Forest Code & Setup

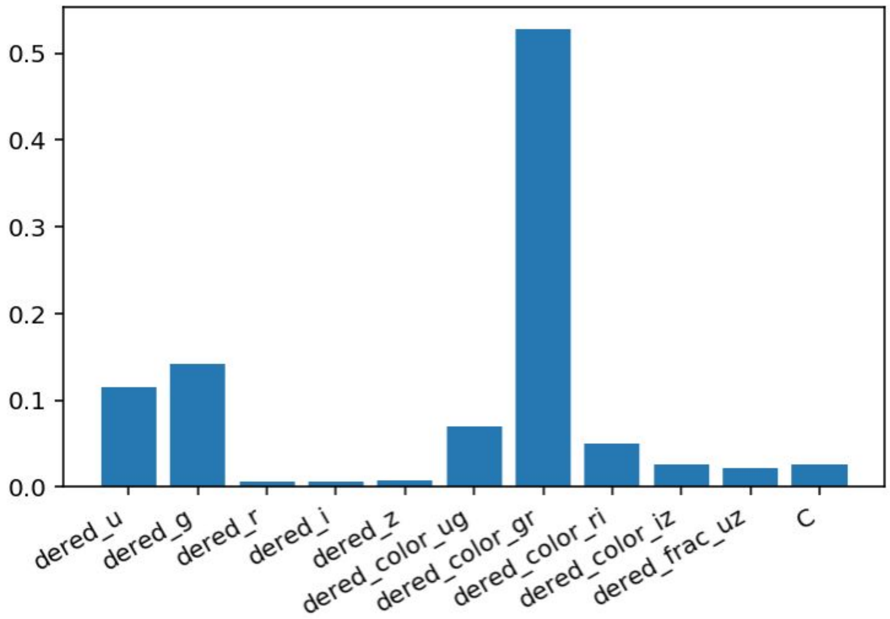
```python
def compute_photoz_forest(depth):
    rms_test = np.zeros(len(depth))
    rms_train = np.zeros(len(depth))
    i_best = 0
    z_fit_best = None

    for i, d in enumerate(depth):

        ### YOU CAN CHANGE N_ESTIMATORS IF YOU LIKE
        clf = RandomForestRegressor(n_estimators=10,
                                    max_depth=d, random_state=0)
        clf.fit(mag_train, z_train)

        z_fit_train = clf.predict(mag_train)
        z_fit = clf.predict(mag_test)
        rms_train[i] = np.mean(np.sqrt(((z_fit_train - z_train)/(1+z_fit_train)) ** 2))
        rms_test[i] = np.mean(np.sqrt(((z_fit - z_test)/(1+z_fit)) ** 2))

        if rms_test[i] <= rms_test[i_best]:
            i_best = i
            z_fit_best = z_fit

    return rms_test, rms_train, i_best, z_fit_best, clf

depth = np.arange(1, 25)
rms_test, rms_train, i_best, z_fit_best, clf = compute_photoz_forest(depth)
best_depth = depth[i_best]
```

RFR in action



RFR: Distributions & Feature importance





Very few predicted redshifts above z = 0.35 ...

Artificial Neural Network: Multi Layer Perceptron

Input layer: photometric features

How each neuron works?

$y = f\,(w \cdot x + b)$

Using dropout to improve robustness

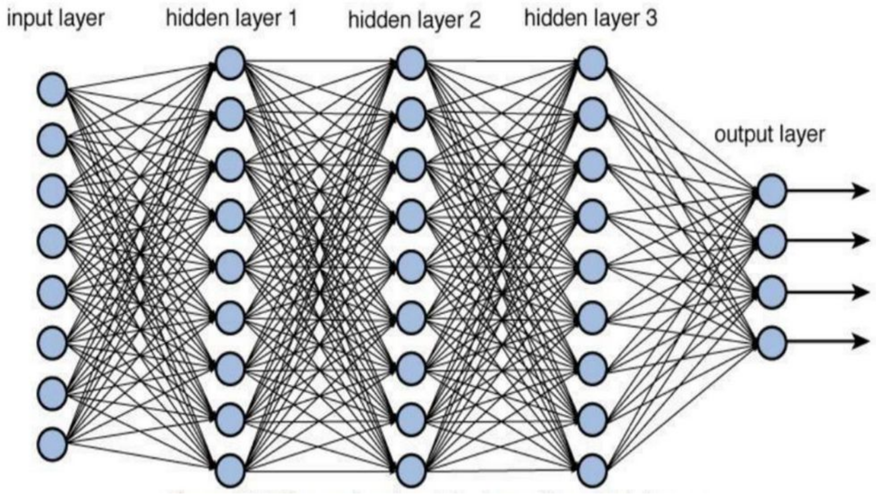
Targeted outputs: categories over $0<z<0.4$

output activation: softmax

loss function: categorical cross-entropy

Training w and b to minimize loss

optimizer: adam



Keras code

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.optimizers import Adam

def MLP(Hlayer,Hneuron,Drate):
    model = Sequential(name=f'{Hlayer}_hidden_layer')
    model.add(Dense(Hneuron, activation='relu', input_dim=11,name=f'hidden_layer_1'))
    model.add(Dropout(Drate,name=f'dropout1_{Drate}'))
    for i in range(Hlayer-1):
        model.add(Dense(Hneuron, activation='relu',name=f'hidden_layer_{i+2}'))
        model.add(Dropout(Drate,name=f'dropout{i+2}_{Drate}'))

    model.add(Dense(180, activation='softmax',name=f'output_layer'))
    return model

filepath='MLP_PDF_MODEL.hdf5'
ModelCheckpointCB = ModelCheckpoint(filepath, monitor='val_loss', verbose=1,
                    save_best_only=True, save_weights_only=False, mode='auto')

model = MLP(3,30,0.1)
adam = Adam(lr=1e-3)
model.compile(optimizer=adam, loss='categorical_crossentropy')
history = model.fit(x=x_train,y=y_train,batch_size=300,epochs=30,verbose=1,
                    validation_data=(x_val,y_val),callbacks=[ModelCheckpointCB])
```

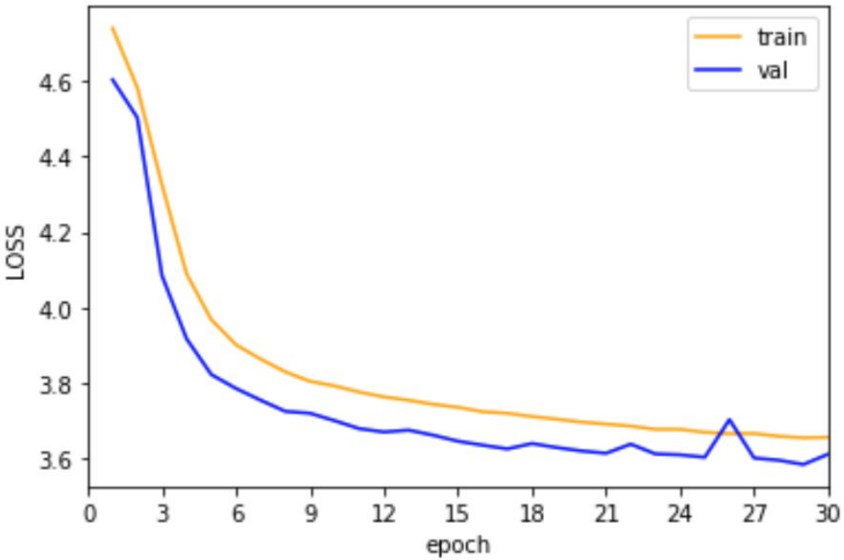
Summary of the Network

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
hidden_layer_1 (Dense)       (None, 30)                360
_________________________________________________________________
dropout1_0.1 (Dropout)       (None, 30)                0
_________________________________________________________________
hidden_layer_2 (Dense)       (None, 30)                930
_________________________________________________________________
dropout2_0.1 (Dropout)       (None, 30)                0
_________________________________________________________________
hidden_layer_3 (Dense)       (None, 30)                930
_________________________________________________________________
dropout3_0.1 (Dropout)       (None, 30)                0
_________________________________________________________________
output_layer (Dense)         (None, 180)               5580
=================================================================
Total params: 7,800
Trainable params: 7,800
Non-trainable params: 0
```

Hyperparameters and Performance Optimization

Besides w and b, we have some hyperparameters to be optimized:

- # of hidden layer
- # of hidden neuron
- learning rate
- training batch size
- training epoch

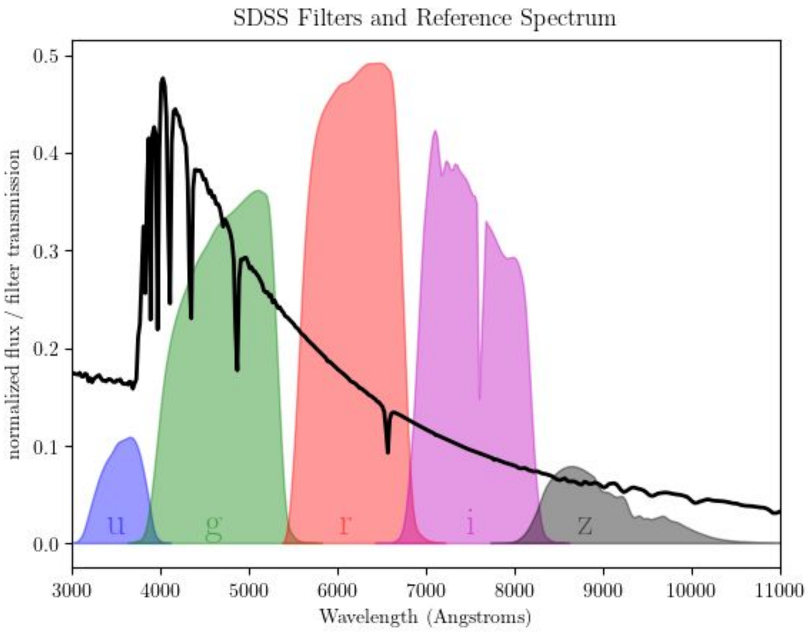


Performance Optimization

using validation set to assess performance at each epoch

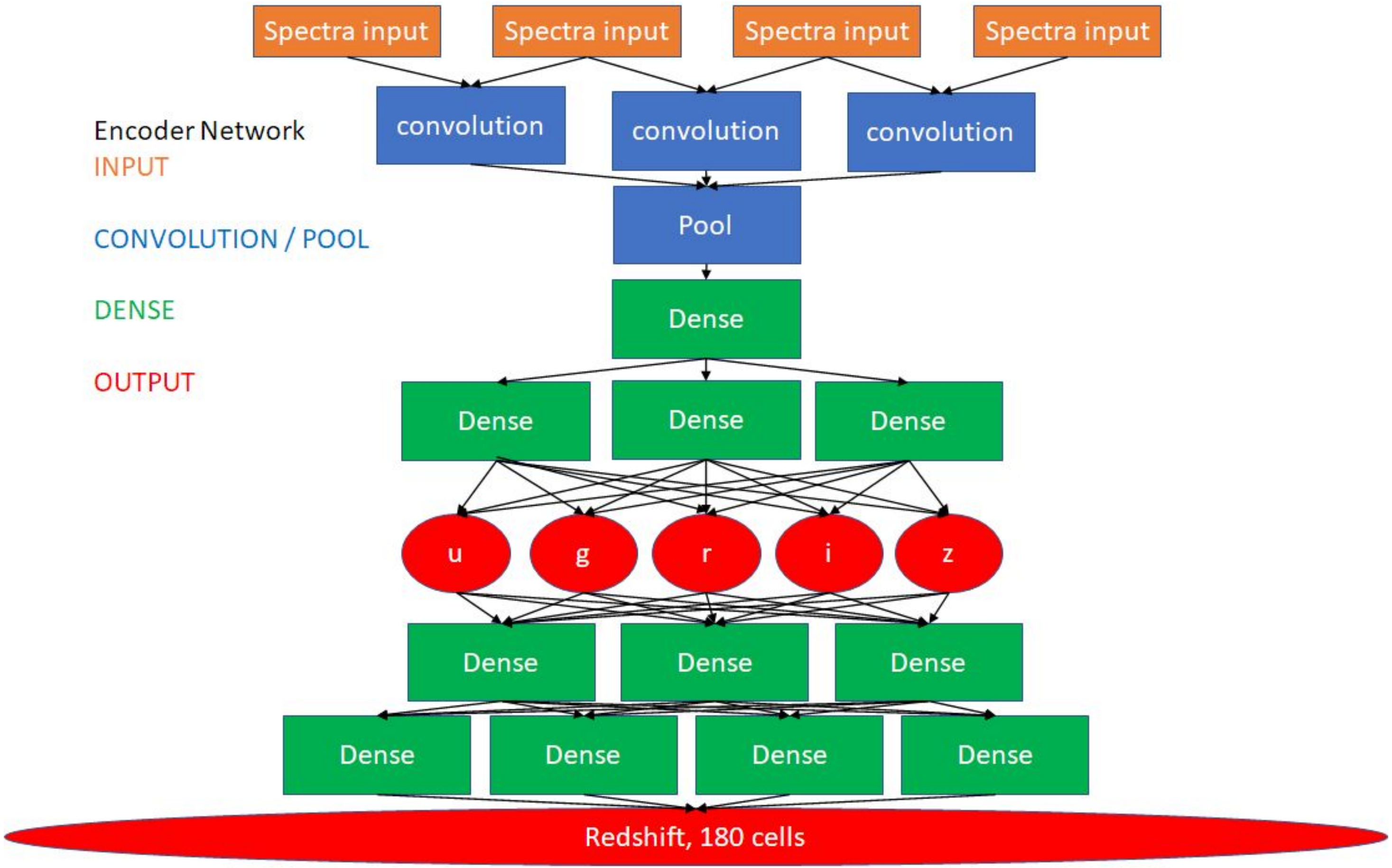
Spectral Encoder Intuition

In Photometric redshifts problem we ask an algorithm to learn a mapping from some feature space to Z space. We the researchers know one way to think about this problem is sliding the spectra over these filters and convolving them, etc. But the algorithm is just fed values without reference to this underlying structure



I wanted to know if there was a way to communicate that underlying structure back to my model, and if that would help in predicting redshifts

Spectral Encoder



Spectral Encoder Architecture

```
def encoder_model():
    Input = keras.layers.Input((3800,1),)
    conv1 = keras.layers.Conv1D(filters=32,kernel_size=7,dilation_rate=1)(Input)
    pool1 = keras.layers.MaxPooling1D(pool_size=2)(conv1)
    conv2 = keras.layers.Conv1D(filters=64,kernel_size=3,dilation_rate=1)(conv1)
    pool2 = keras.layers.MaxPooling1D(pool_size=2)(conv2)
    edense1 = keras.layers.Dense(252,activation=keras.activations.relu)(pool2)
    edense2 = keras.layers.Dense(128,activation=keras.activations.relu)(edense1)
    edense3 = keras.layers.Dense(5,activation=keras.activations.linear,name='output1')(edense2)

    dense1 = keras.layers.Dense(45,activation=keras.activations.relu)(edense3)
    drop1 = keras.layers.Dropout(0.05)(dense1)

    dense2 = keras.layers.Dense(45,activation=keras.activations.relu)(drop1)
    drop2 = keras.layers.Dropout(0.05)(dense2) #0.1

    dense3 = keras.layers.Dense(45,activation=keras.activations.relu)(drop2)
    drop3 = keras.layers.Dropout(0.05)(dense3)

    dense4 = keras.layers.Dense(45,activation=keras.activations.relu)(drop3)
    drop4 = keras.layers.Dropout(0.05)(dense4)#0.1

    dense5 = keras.layers.Dense(180,activation=keras.activations.softmax,name='output2')(drop4)

    model = keras.Model(inputs=[Input],outputs=[edense3,dense5])
    return model
```

Keras with Multiple Outputs

```
losses = {
    "output1": "MSE",
    "output2": "categorical_crossentropy",
}

lossWeights = {"output1": 1.0, "output2": 1.0}

encoder.compile(optimizer=adam_e, loss=losses, loss_weights=lossWeights)
```

With two outputs we have to set the network up for two losses, and we can tell the network how to weigh these outputs relative to one another

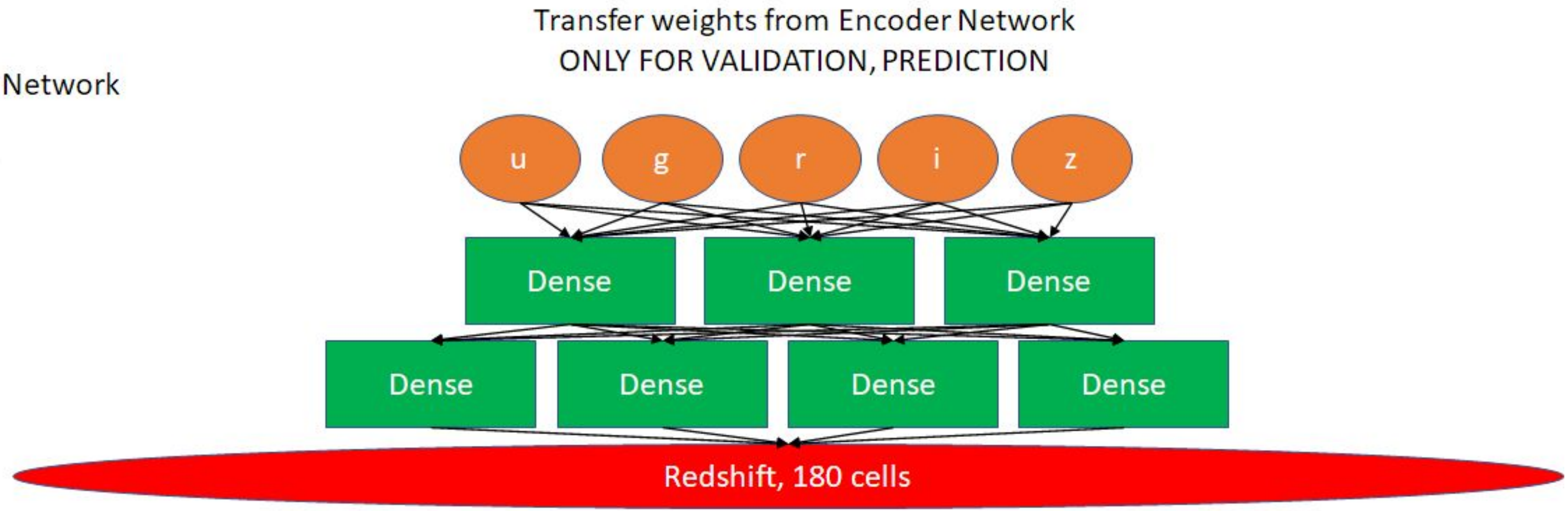
Spectral Encoder: Predictions

Decoder Network

OUTPUT

INPUT

DENSE



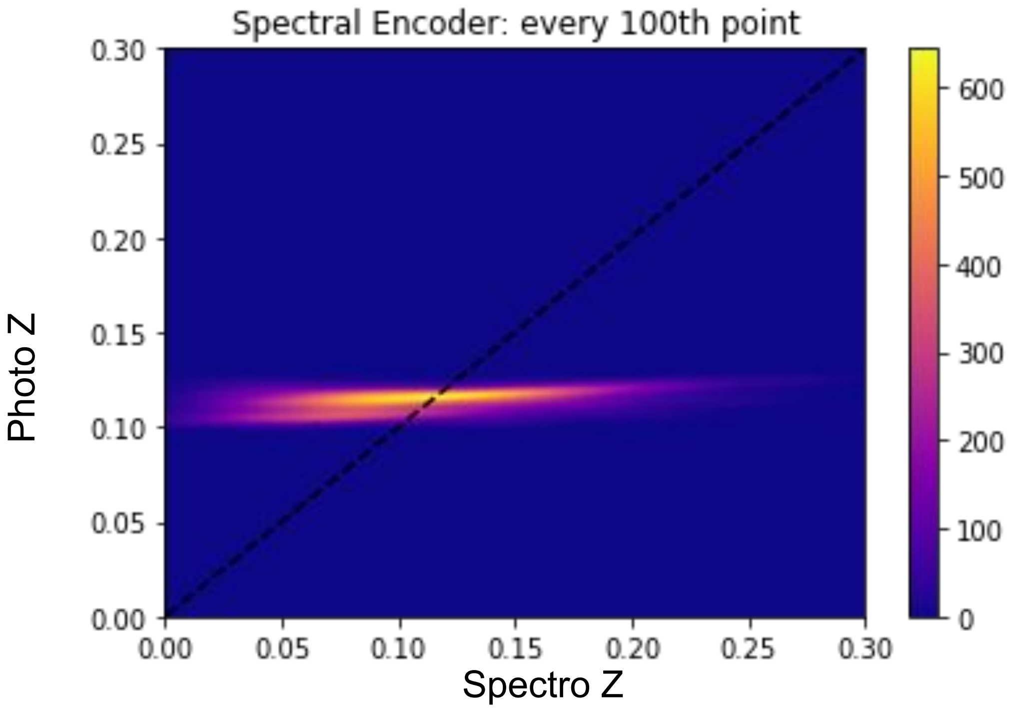
Spectral Encoder: Conclusion

Doesn't Work:

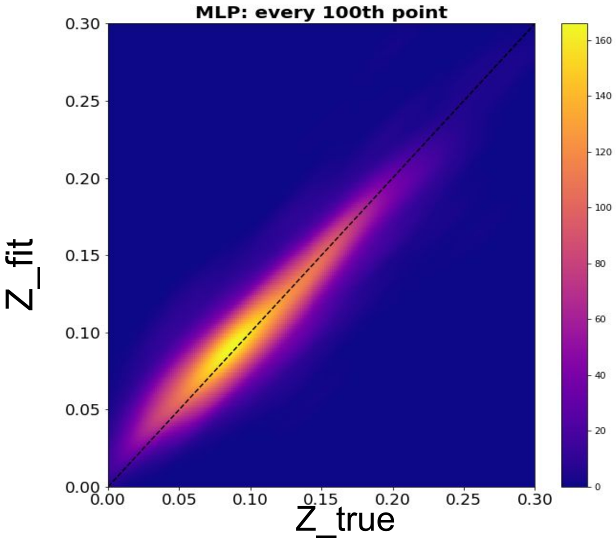
The Model's architecture can't translate the mapping of spectra onto magnitudes into anything meaningful. **Of course it can't there are no free parameters for it to do that.**

It's also possible that the model chose to learn a mapping of spectra onto Z using 5 parameters, it's just that those 5 parameters are not tightly constrained enough to be magnitudes.

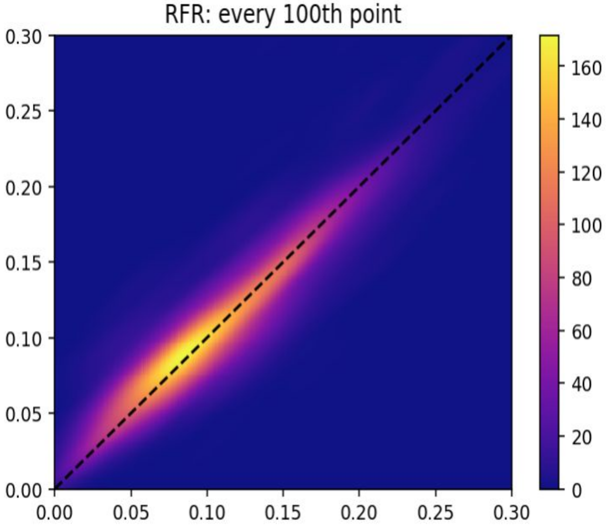
Also possible that my intuition is just dead wrong



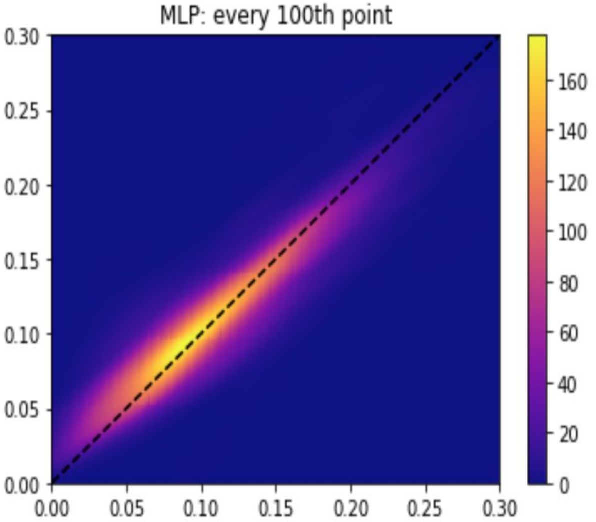
K-Nearest Neighbors



RF



ANN



	MAD	Bias	Outlier Fraction η
KNN	0.0168	-0.0006	0.0017
RF	0.0147	-0.0009	0.0025
ANN	0.0158	-0.0014	0.0024
Spectral Encoder	0.048	-0.0047	0.0

Conclusion

Accurate, precise, & fast Photo-Z algorithms will become ever more important with future wide-deep missions like LSST.

We have shown the researcher’s choice of algorithm affects their results; and since there is no one metric that must be optimized for any Photo-Z application, each has merit. (See S.J. Schmidt, A.I. Malz, *et. al.,* 2020)

We propose a modified version of K-Nearest Neighbors, called...

Gautham Nearest Neighbors (GNN)