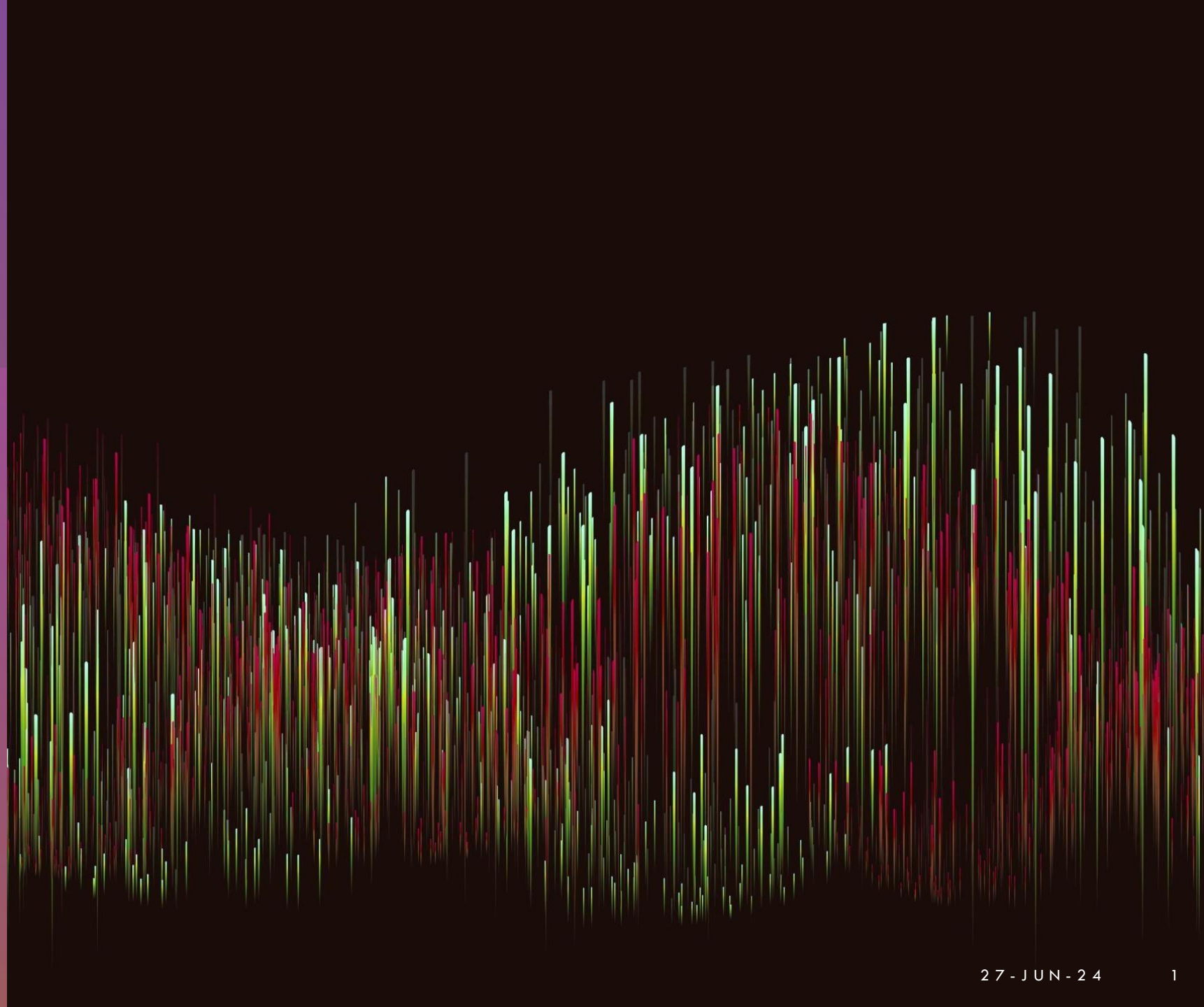


MALTE BRUHN
SASCHA MAHMOOD

X-RAY IMAGE SEGMENTATION FOR FRACTURE PREDICTION



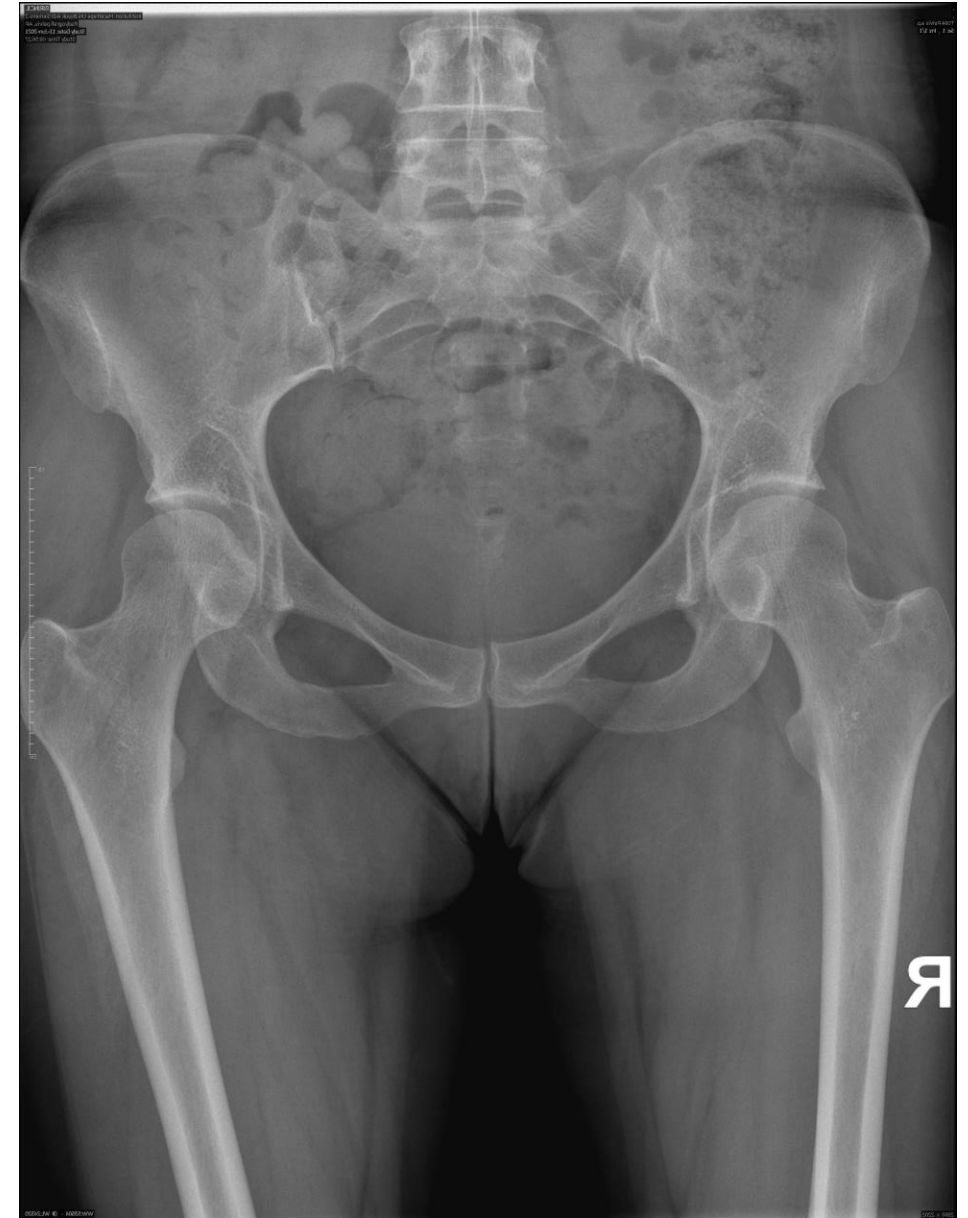
AGENDA

- Introduction
- Data
- Models
- Baseline → Dice Metric
- Final Model → Augmentation
- Final Model → Hyperparameter Optimization



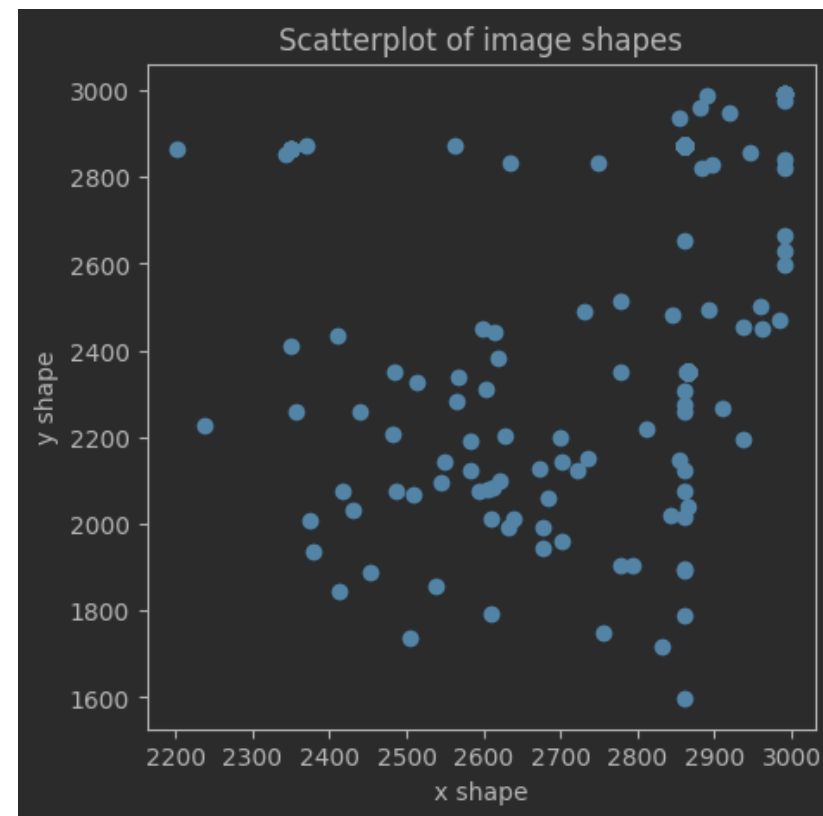
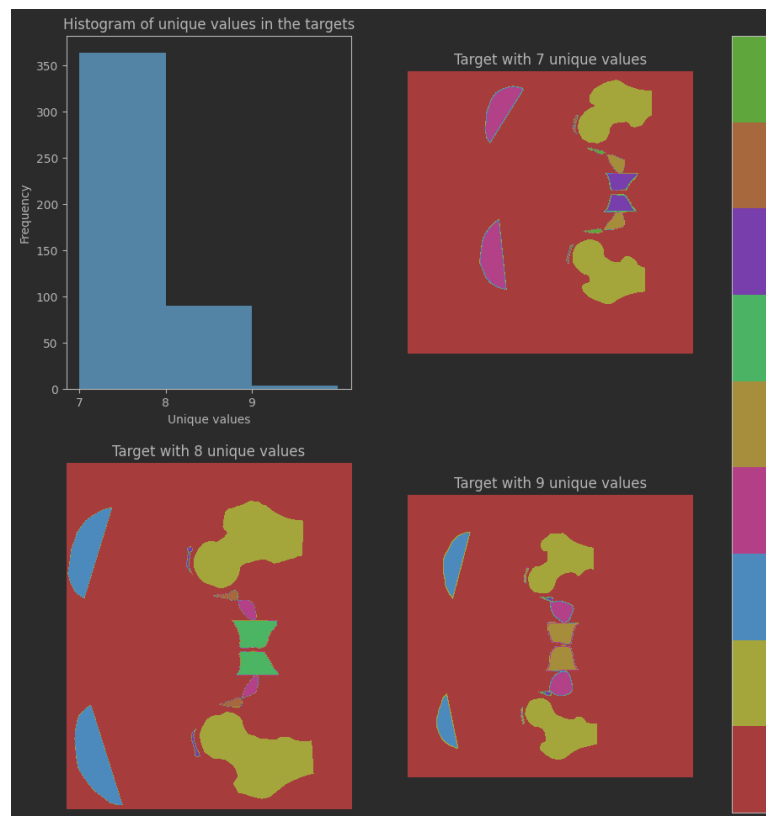
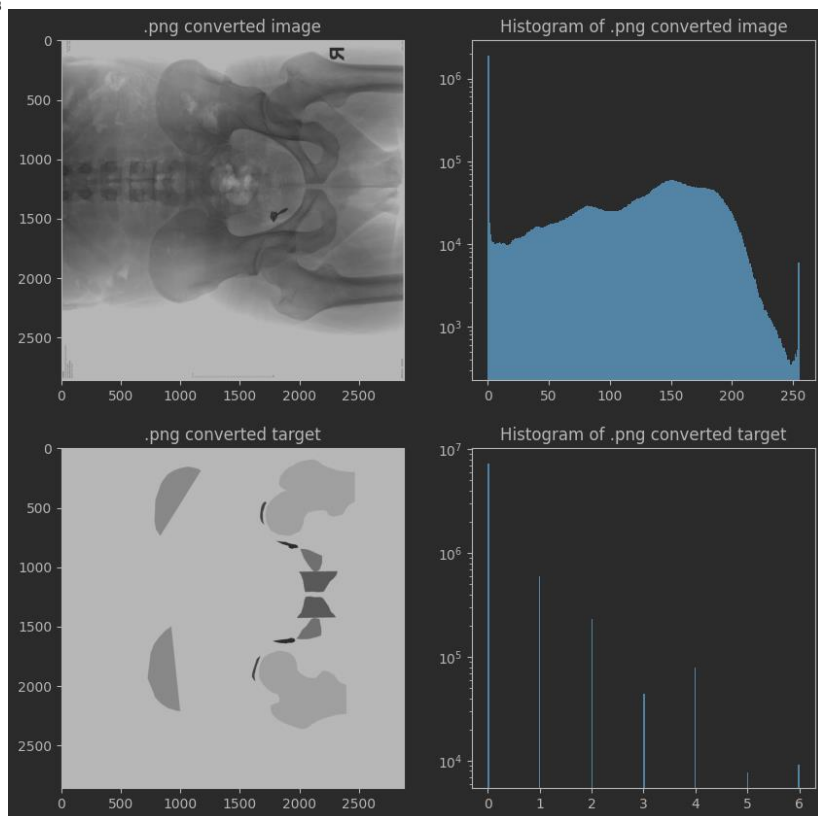
INTRODUCTION

- Eventual goal:
 - Determining the likelihood of fractures of the proximal femur based on x-ray images
- Our goal:
 - Segment the proximal femur in x-ray images



DATA

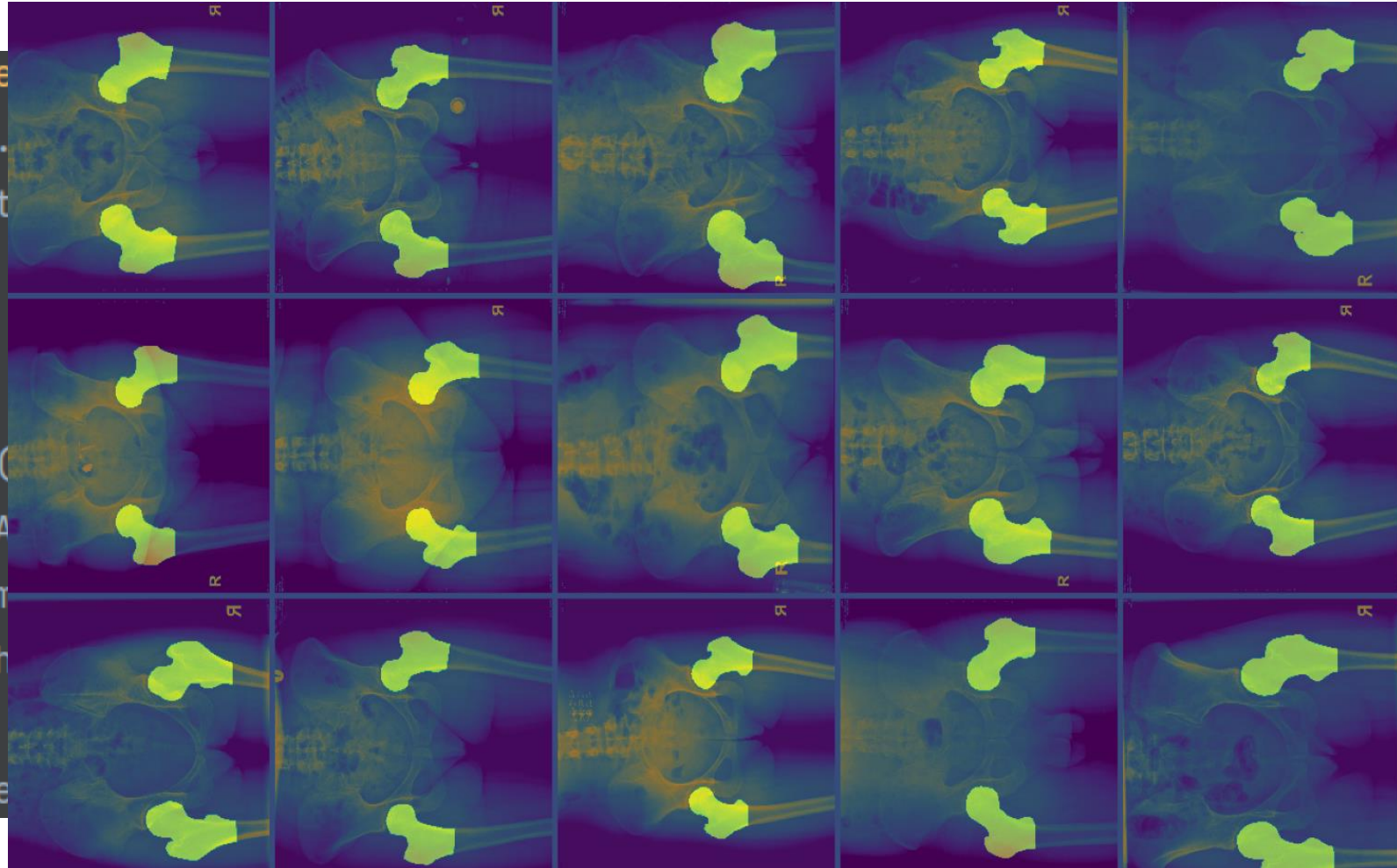
- Data from website
 - <https://universe.roboflow.com/ks-fsm9o/pelvis-ap-x-ray/dataset/2>
 - Already split into training, validation and test set
 - BUT: in NIFTI format → conversion needed



DATA

- Data loader based on the `tf.data.Dataset` class
- File names were supplied and then replaced by the files via a mapping

```
def create_dataset  
    dataset = tf.  
    dataset = dat  
    batches = (  
        dataset  
        .cache()  
        .shuffle(  
        .batch(BA  
        .map(Augm  
        .prefetch  
    )  
    return batche
```



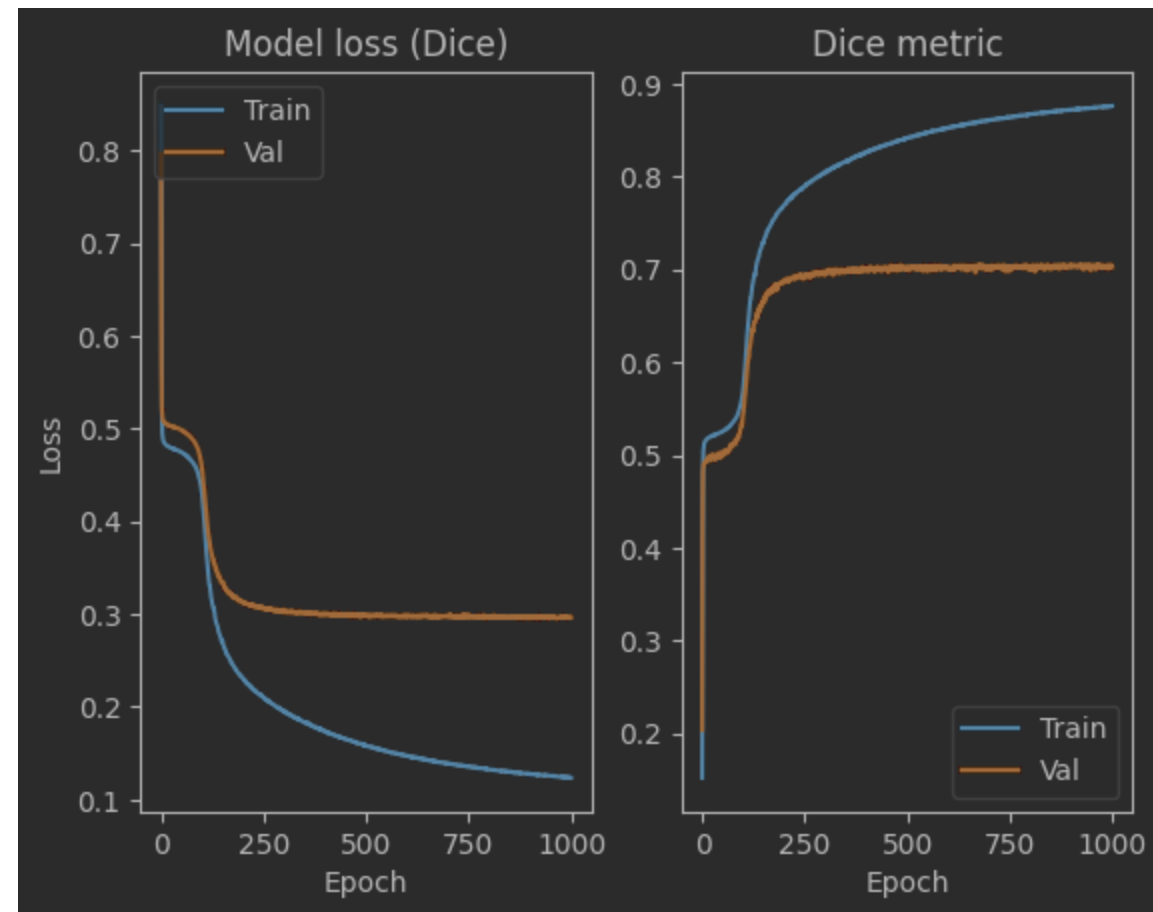
ental.AUTOTUNE)

BASELINE / DICE METRIC

- Simple model
- Only one two densely connected layers after a flattening layer and a reshaping back to an image

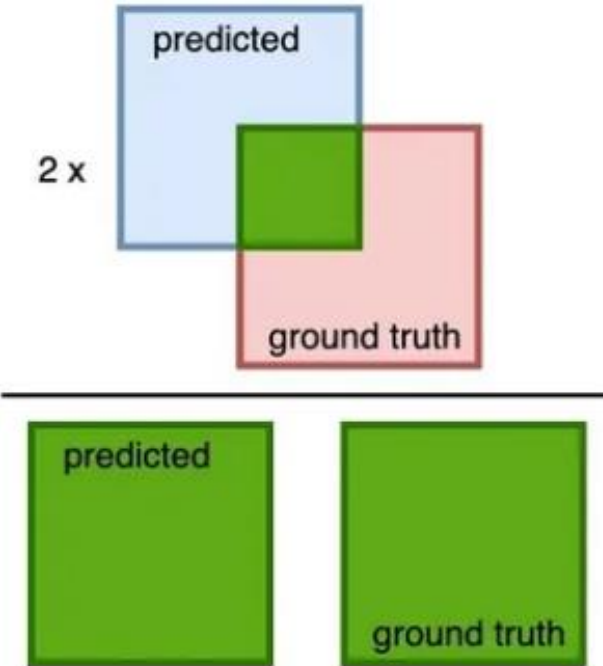
```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=[*IMSIZE, 1]),
    tf.keras.layers.Dense(1024, activation='relu'),
    tf.keras.layers.Dense(np.prod(IMSIZE), activation='sigmoid'),
    tf.keras.layers.Reshape((*IMSIZE, 1))
])

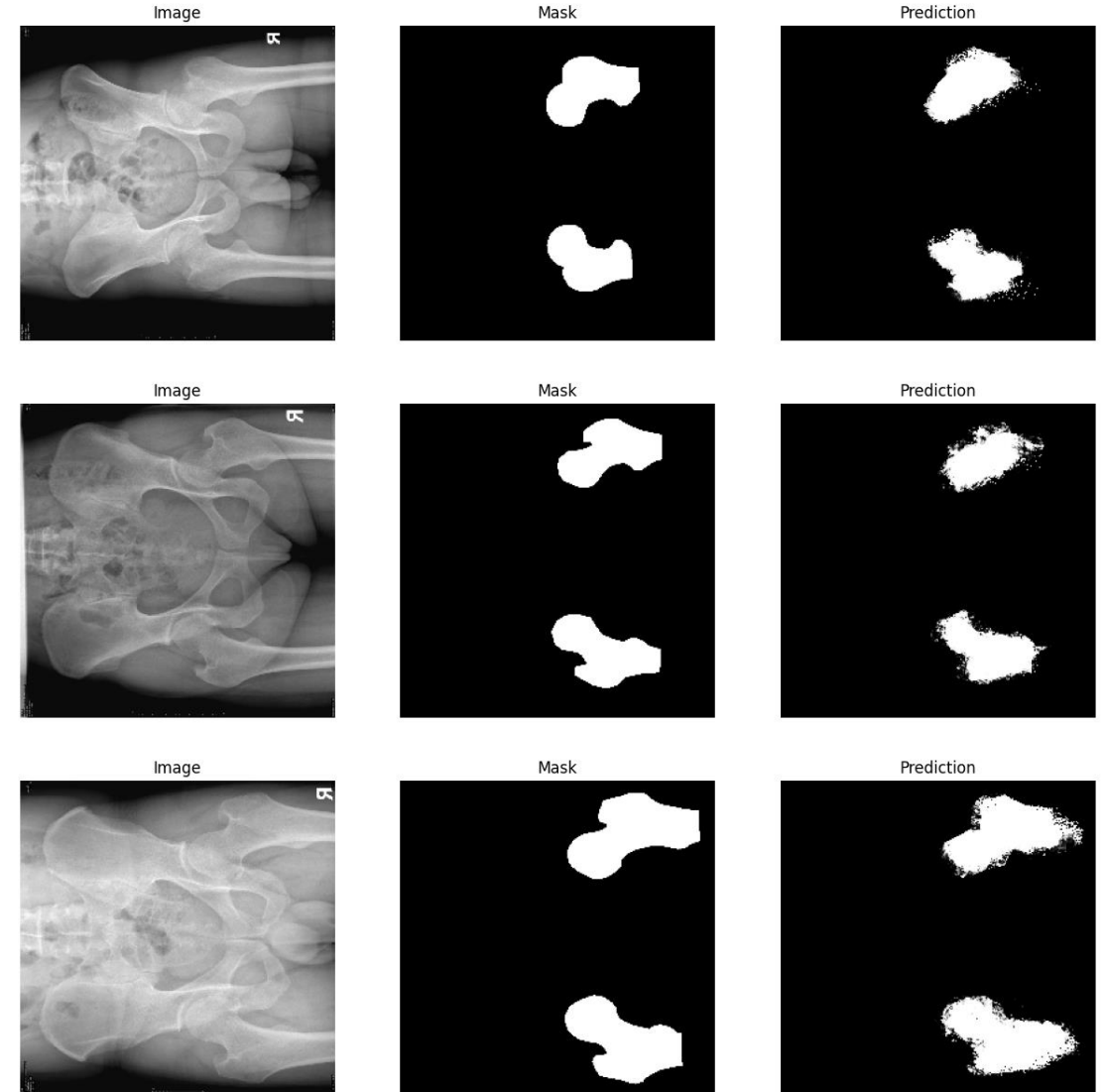
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.5e-4),
    loss=dice,
    metrics=[dice_metric])
```



BASLINE / DICE METRIC

- Baseline dice coefficient: 0.705
- Dice loss vs. Binary Crossentropy
- Dice coefficient vs. Accuracy

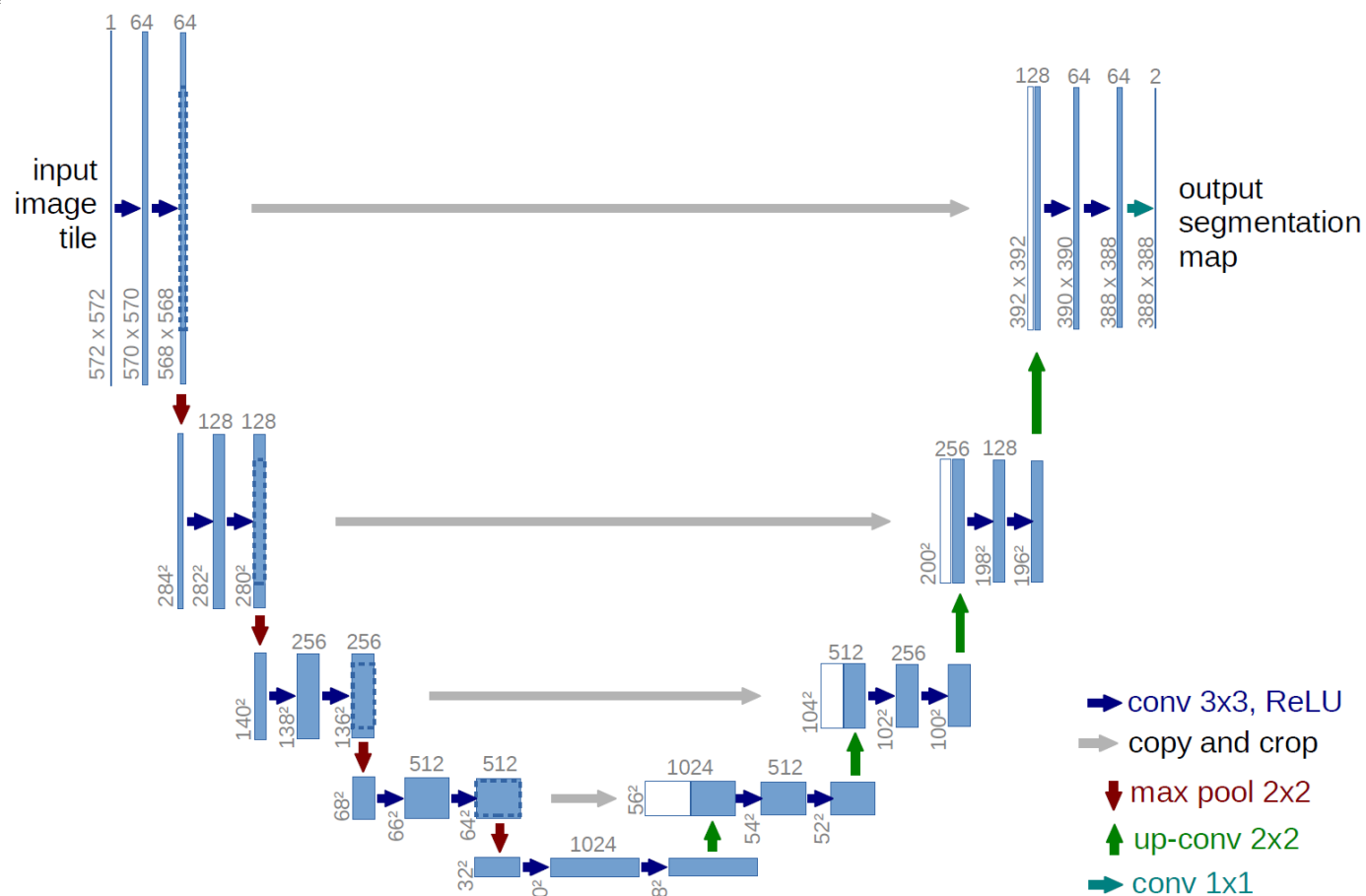
$$\text{Dice coefficient} = \frac{2 \times \text{area of overlapped (green)}}{\text{total area (green)}} = \frac{2 \times \text{predicted} \cap \text{ground truth}}{\text{predicted} \cup \text{ground truth}}$$




<https://cvinvolution.medium.com/dice-loss-in-medical-image-segmentation-d0e476eb486>, visited 6/6/24

FINAL MODEL / ARCHITECTURE

- U-Net architecture



```
def get_model():
    in1 = Input(shape=(*IMSIZE, 1))

    conv1 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(in1)
    conv1 = Dropout(0.2)(conv1)
    conv1 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(conv1)
    pool1 = MaxPooling2D((2, 2))(conv1)

    conv2 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(pool1)
    conv2 = Dropout(0.2)(conv2)
    conv2 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(conv2)
    pool2 = MaxPooling2D((2, 2))(conv2)

    conv3 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(pool2)
    conv3 = Dropout(0.2)(conv3)
    conv3 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(conv3)
    pool3 = MaxPooling2D((2, 2))(conv3)

    conv4 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(pool3)
    conv4 = Dropout(0.2)(conv4)
    conv4 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(conv4)

    up1 = concatenate([UpSampling2D((2, 2))(conv4), conv3], axis=-1)
    conv5 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(up1)
    conv5 = Dropout(0.2)(conv5)
    conv5 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(conv5)

    up2 = concatenate([UpSampling2D((2, 2))(conv5), conv2], axis=-1)
    conv6 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(up2)
    conv6 = Dropout(0.2)(conv6)
    conv6 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(conv6)

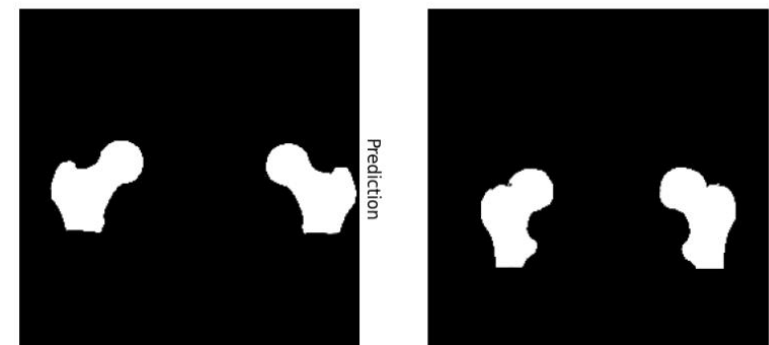
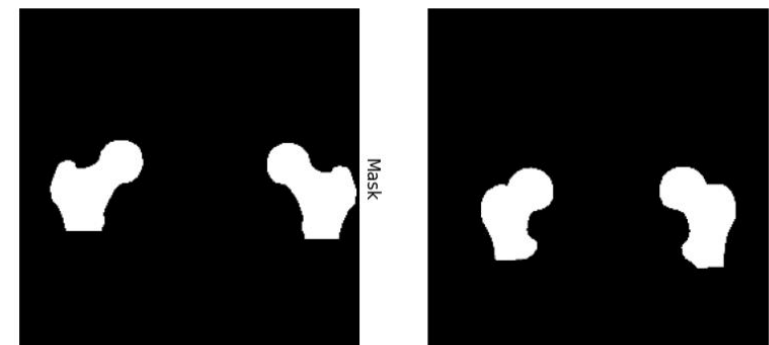
    up2 = concatenate([UpSampling2D((2, 2))(conv6), conv1], axis=-1)
    conv7 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(up2)
    conv7 = Dropout(0.2)(conv7)
    conv7 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(conv7)
    segmentation = Conv2D(1, (1, 1), activation='sigmoid', name='seg')(conv7)

    model = Model(inputs=[in1], outputs=[segmentation])

    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
                  loss=dice_coef_loss,
                  metrics=['accuracy', dice_coef])
```


FINAL MODEL / AUGMENTATION

- Usual augmentation techniques (rotation, flipping, zoom)
 - Longer training
 - No increase in model performance
- → Highly structured dataset
 - All images oriented the same way
 - Femur always in roughly the same location
 - Always exactly two femoral heads
- Instead, dropout was used for model regulation



FINAL MODEL / HYPER PARAMETER OPTIMIZATION

using learning rate: 0.0003

```
determine_learning_rate = True
learning_rate_results = {}

if determine_learning_rate:
    # determine learning rate
    for learning_rate in [10 ** -i for i in range(3,6)]:
        for multiplier in range(10):
            lr = learning_rate * (multiplier + 1)
            model = unet()
            history = model.fit(train_batches,
                                validation_data=val_batches,
                                epochs=10,
                                verbose=2)
            learning_rate_results[lr] = max(history.history['val_dice_metric'])
```

FINAL MODEL / HYPER PARAMETER OPTIMIZATION

```
parameters={  
    'init': ['he_normal', 'glorot_uniform'],  
    'dropout_rate': [0.2, 0.3, 0.5],  
    'adam_beta1': [i / 10 for i in range(1,10)],  
    'adam_beta2': [0.990 + 10 * i / 10000 for i in range(1,10)]  
}
```

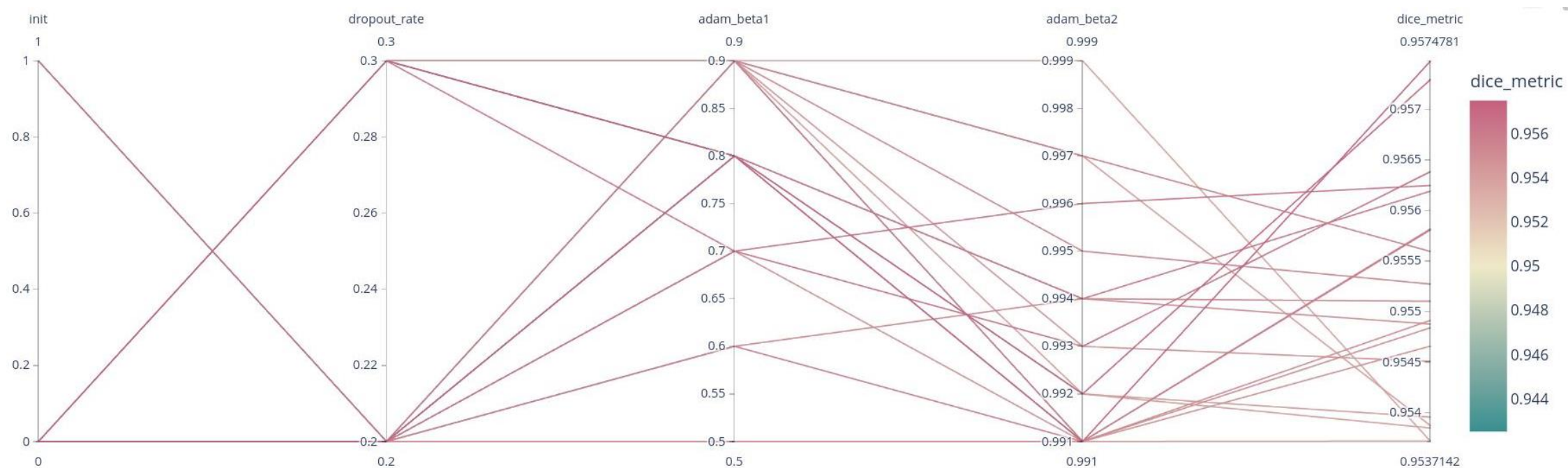
```
('init', ['he_normal', 'glorot_uniform'])  
('dropout_rate', [0.2, 0.3, 0.5])  
('adam_beta1', [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])  
('adam_beta2', [0.991, 0.992, 0.993, 0.994, 0.995, 0.996, 0.997, 0.998, 0.999])
```

486 different combinations taking about 20h to train

FINAL MODEL / HYPER PARAMETER OPTIMIZATION

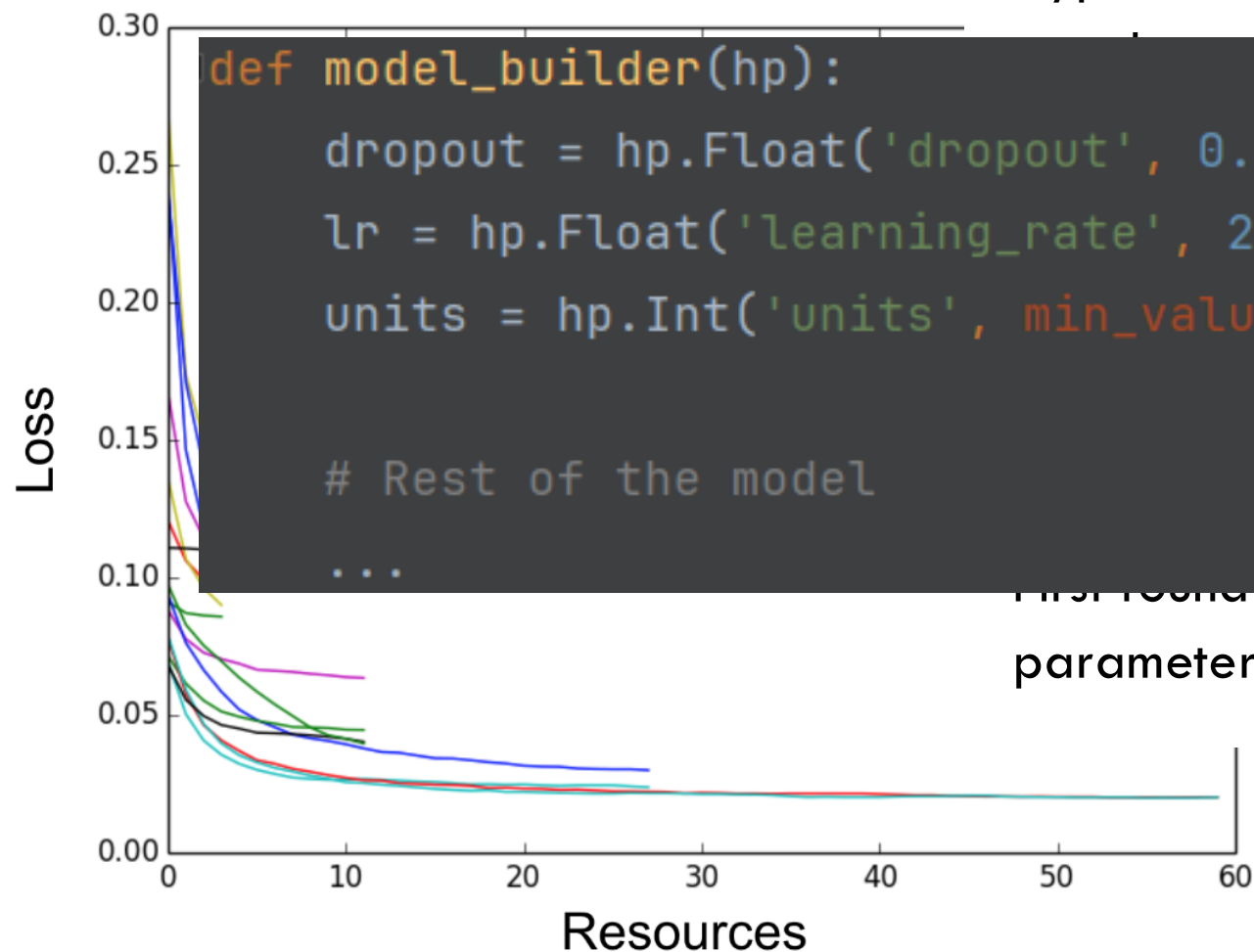
```
{'init': 'glorot_uniform', 'dropout_rate': 0.3, 'adam_beta1': 0.8, 'adam_beta2': 0.991, 'lr': 0.00030000000000000003} - 0.9574781060218811
{'init': 'glorot_uniform', 'dropout_rate': 0.2, 'adam_beta1': 0.8, 'adam_beta2': 0.992, 'lr': 0.00030000000000000003} - 0.9572945237159729
{'init': 'he_normal', 'dropout_rate': 0.2, 'adam_beta1': 0.7, 'adam_beta2': 0.993, 'lr': 0.00030000000000000003} - 0.9563837647438049
{'init': 'glorot_uniform', 'dropout_rate': 0.3, 'adam_beta1': 0.7, 'adam_beta2': 0.996, 'lr': 0.00030000000000000003} - 0.9562456011772156
{'init': 'glorot_uniform', 'dropout_rate': 0.3, 'adam_beta1': 0.8, 'adam_beta2': 0.994, 'lr': 0.00030000000000000003} - 0.9561899304389954
{'init': 'he_normal', 'dropout_rate': 0.2, 'adam_beta1': 0.6, 'adam_beta2': 0.991, 'lr': 0.00030000000000000003} - 0.9558175206184387
{'init': 'glorot_uniform', 'dropout_rate': 0.2, 'adam_beta1': 0.9, 'adam_beta2': 0.991, 'lr': 0.00030000000000000003} - 0.955807626247406
{'init': 'glorot_uniform', 'dropout_rate': 0.2, 'adam_beta1': 0.9, 'adam_beta2': 0.997, 'lr': 0.00030000000000000003} - 0.955592155456543
{'init': 'glorot_uniform', 'dropout_rate': 0.2, 'adam_beta1': 0.9, 'adam_beta2': 0.995, 'lr': 0.00030000000000000003} - 0.9552703499794006
{'init': 'glorot_uniform', 'dropout_rate': 0.2, 'adam_beta1': 0.6, 'adam_beta2': 0.994, 'lr': 0.00030000000000000003} - 0.9551003575325012
{'init': 'glorot_uniform', 'dropout_rate': 0.2, 'adam_beta1': 0.8, 'adam_beta2': 0.991, 'lr': 0.00030000000000000003} - 0.9549126029014587
{'init': 'glorot_uniform', 'dropout_rate': 0.2, 'adam_beta1': 0.8, 'adam_beta2': 0.994, 'lr': 0.00030000000000000003} - 0.9548751711845398
{'init': 'he_normal', 'dropout_rate': 0.2, 'adam_beta1': 0.5, 'adam_beta2': 0.991, 'lr': 0.00030000000000000003} - 0.9548394083976746
{'init': 'he_normal', 'dropout_rate': 0.2, 'adam_beta1': 0.7, 'adam_beta2': 0.991, 'lr': 0.00030000000000000003} - 0.9546578526496887
{'init': 'glorot_uniform', 'dropout_rate': 0.3, 'adam_beta1': 0.9, 'adam_beta2': 0.993, 'lr': 0.00030000000000000003} - 0.9545062184333801
{'init': 'he_normal', 'dropout_rate': 0.2, 'adam_beta1': 0.9, 'adam_beta2': 0.992, 'lr': 0.00030000000000000003} - 0.9539551734924316
{'init': 'glorot_uniform', 'dropout_rate': 0.3, 'adam_beta1': 0.9, 'adam_beta2': 0.997, 'lr': 0.00030000000000000003} - 0.9538684487342834
{'init': 'glorot_uniform', 'dropout_rate': 0.3, 'adam_beta1': 0.8, 'adam_beta2': 0.992, 'lr': 0.00030000000000000003} - 0.9538471102714539
{'init': 'glorot_uniform', 'dropout_rate': 0.3, 'adam_beta1': 0.9, 'adam_beta2': 0.991, 'lr': 0.00030000000000000003} - 0.9537174701690674
{'init': 'glorot_uniform', 'dropout_rate': 0.2, 'adam_beta1': 0.9, 'adam_beta2': 0.999, 'lr': 0.00030000000000000003} - 0.9537141919136047
```


FINAL MODEL / HYPER PARAMETER OPTIMIZATION



HYPERPARAMETER OPTIMIZATION - HYPERBAND

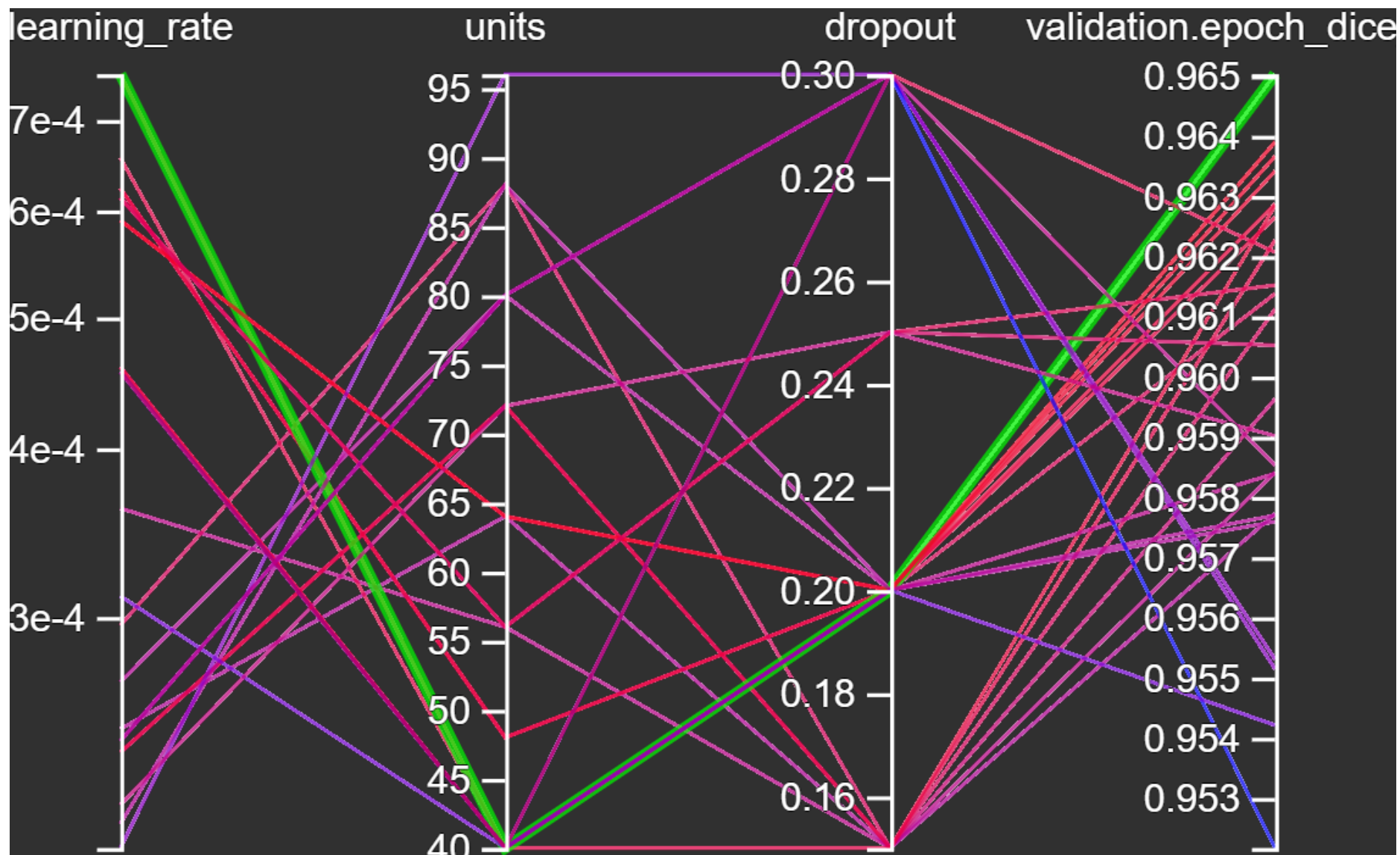
- Hyperband tuner allocates a computation budget per



parameter space

Li, Lisha et al. "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization." *J. Mach. Learn. Res.* 18 (2016): 185:1-185:52.

RESULTS



FINAL RESULT

- Baseline score: 0.705
- Score with manually optimized U-Net: 0.962
- Score with Hyperband tuning: 0.965
- Only marginal improvements over the (heavily optimized) base U-Net (0.3 %)
- But overall large improvements over the simple baseline model (36.9 %)

