Story Squad Experiments

This document can be found online @ https://honored-mandolin-bf6.notion.site/Story-Squad-Experiments-0ccabcb3e3d44b4d8ee3057c68ad1830

Research question:

Is it possible to design a model to replace user input in story_photo_transformer.py and result in comparable transformed image quality? is it possible using the synthetic dataset provided by generate.py and samples of user input provided by story_photo_transformer.py to generate coordinates with less than 8pix mean average error?

Assumptions:

(these should be tested but time does not allow)

1. A 3 layer fully connected network of 32 neurons each will not provide a low enough MAE
2. training on less than 1000 samples without synthetic data on any architecture will not provide low enough MAE

Hypothesis:

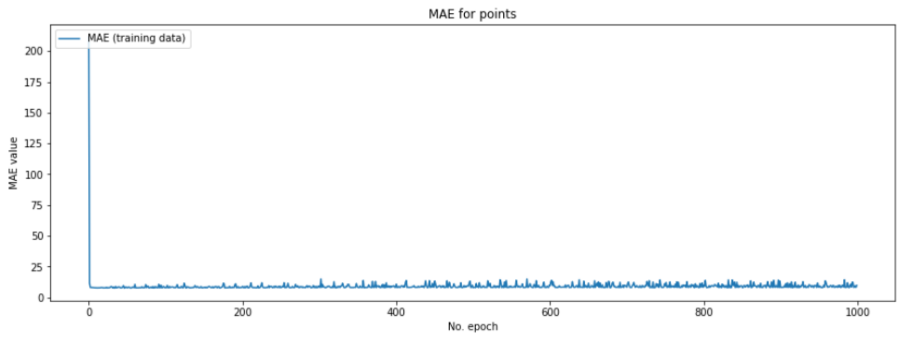
Training a 3 layer fully connected network on synthetic data first will improve MAE on user generated data.

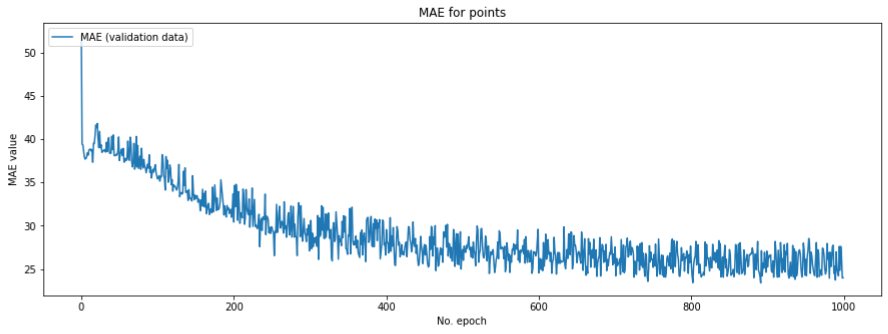
▼ Models (Python code for model generation)

```
c_init = keras.initializers.Constant(.004)
b_init = keras.initializers.Constant(.004)

img_inputs = keras.Input(shape=(256, 256-64, 3))
x = img_inputs
x = layers.Flatten()(x)

x = layers.Dense(32, activation="relu",dtype="float32",kernel_initializer=c_init,bias_initializer=b_init)
(x)
x = layers.Dense(32, activation="relu",dtype="float32",kernel_initializer=c_init,bias_initializer=b_init)
(x)
x = layers.Dense(32, activation="relu",dtype="float32",kernel_initializer=c_init,bias_initializer=b_init)
(x)

outputs = layers.Dense(8,activation="relu",kernel_initializer=c_init,bias_initializer=b_init)(x)

model = keras.Model(inputs=img_inputs, outputs=outputs, name="FC_Model")
model.summary()
model.compile(loss="MAE",optimizer=tf.keras.optimizers.Adam(learning_rate=global_learning_rate))
```

64 nodes per layer

Testing for MAE with the user generated dataset while training on the synthetic dataset shows in all three cases that training the network on the synthetic dataset does lower the MAE. This is actually the case in all of the experiments in this paper. The synthetic data always improves accuracy on the real data.

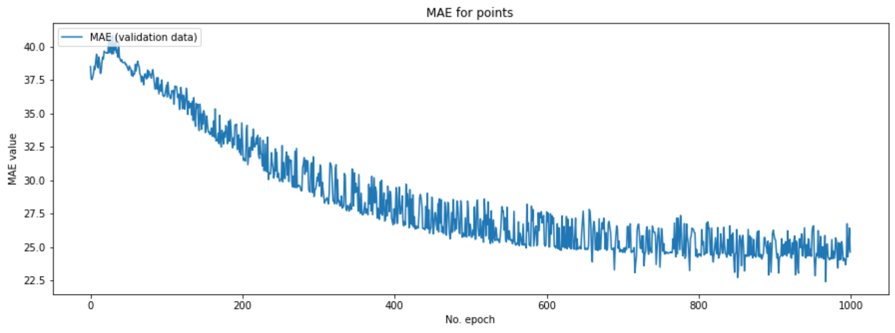




```
The minimum MAE achieved by the training was 7.658749580383301.
The minimum MAE achieved on the real data was 23.40812873840332.
```

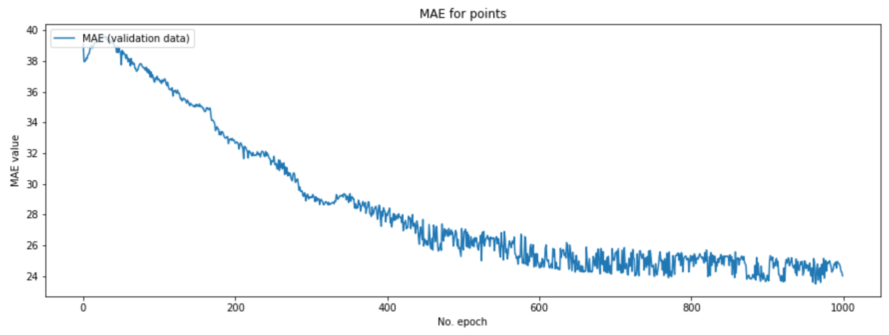
I use the 32 node per layer model as the baseline for the remainder of this paper.

32 Nodes per layer



```
The minimum MAE achieved by the training was 7.689208984375.
The minimum MAE achieved on the real data was 22.393457412719727.
```

16 nodes per layer



```
The minimum MAE achieved by the training was 7.590624809265137.
The minimum MAE achieved on the real data was 23.487966537475586.
```

Hypothesis Conclusion

The graphs show that continued training on the synthetic data improves accuracy on the unseen real data even when no real improvement is seen on the synthetic data. This proves our Hypothesis for this experiment True.

Hypothesis 2:

Adding convolution layers to the beginning of the model will improve MAE to below baseline levels

Definition: C2 Unit, consists of a 2d convolution layer, relu activation and max pooling

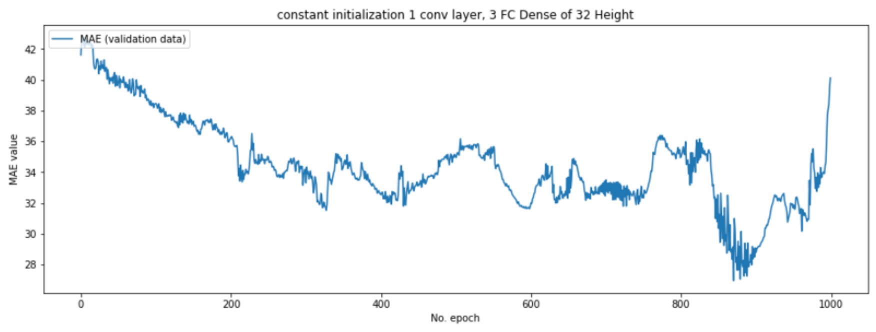
▼ Models (Python code for model generation)

```
def create_experiment_model(num_conv,initialization):
    if initialization == "constant":
        c_init = keras.initializers.Constant(.004)
        b_init = keras.initializers.Constant(.004)
    else:
        c_init=None
        b_init=None

    img_inputs = keras.Input(shape=(256, 256-64, 3))
    x = img_inputs

    for _ in range(int(num_conv)):
        x = layers.Conv2D(32,(3,3),(2,2))(x)
        x = tf.keras.layers.ReLU()(x)
        x = layers.MaxPool2D(strides=(2,2))(x)

    x = layers.Flatten()(x)
    x = layers.Dense(32, activation="relu",dtype="float32",kernel_initializer=c_init,bias_initializer=b_i
nit)(x)
    x = layers.Dense(32, activation="relu",dtype="float32",kernel_initializer=c_init,bias_initializer=b_i
nit)(x)
    x = layers.Dense(32, activation="relu",dtype="float32",kernel_initializer=c_init,bias_initializer=b_i
nit)(x)

    outputs = layers.Dense(8,activation="relu",kernel_initializer=c_init,bias_initializer=b_init)(x)

    model = keras.Model(inputs=img_inputs, outputs=outputs, name="FC_Model")
    model.summary()
    model.compile(loss="MAE",optimizer=tf.keras.optimizers.Adam(learning_rate=global_learning_rate))
    return model
```

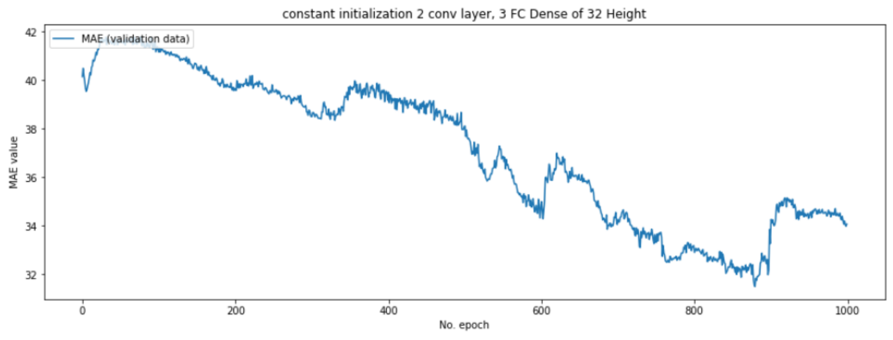
The constant initialization group exhibits erratic results that suggest the optimizer (Adam) is searching for the correct solution.

22.39 MAE is our baseline model's accuracy on the validation set, and this best score is 26.94 on the 1 convolution model.

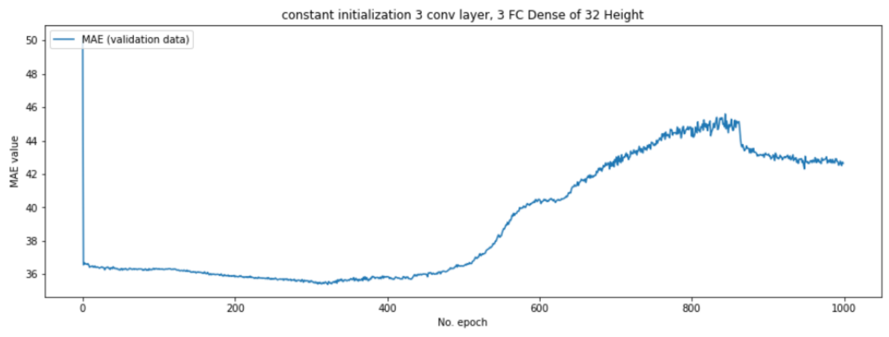
Constant initialization group (Group Q)



```
By epoch 1000 The minimum MAE achieved by the training was 7.710071563720703.
By epoch 1000 The minimum MAE achieved on the real data was 26.942642211914062.
```



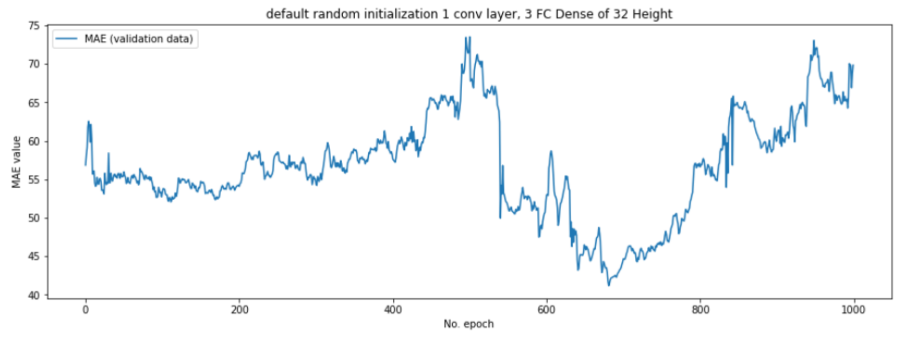
```
By epoch 1000 The minimum MAE achieved by the training was 7.577856063842773.
By epoch 1000 The minimum MAE achieved on the real data was 31.495946884155273.
```



```
By epoch 1000 The minimum MAE achieved by the training was 7.870067119598389.
By epoch 1000 The minimum MAE achieved on the real data was 35.388858795166016.
```

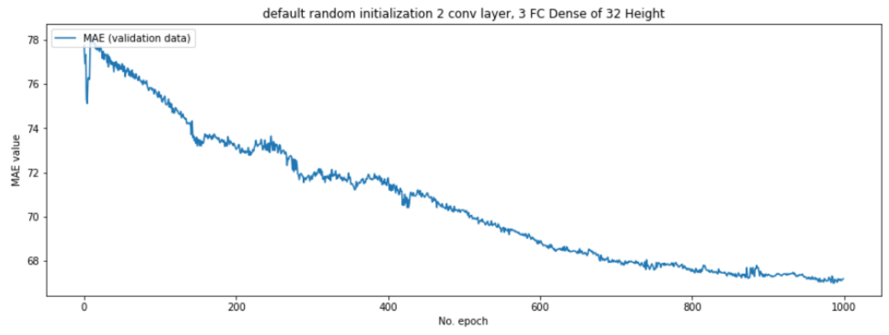
Randomized initialization group (Group R)

The default random initialization group

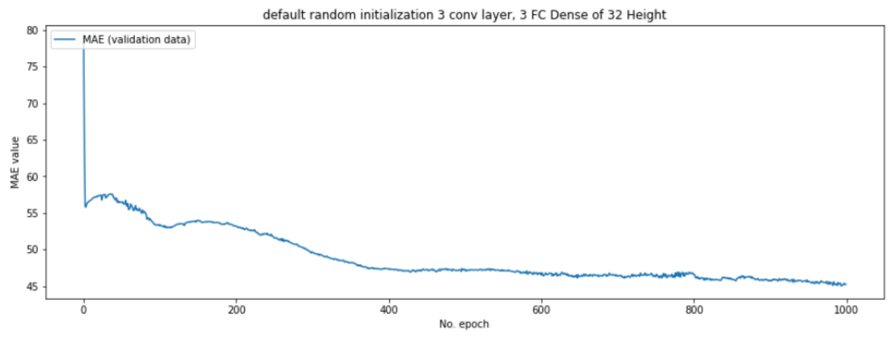


```
By epoch 1000 The minimum MAE achieved by the training was 3.367377996444702.
By epoch 1000 The minimum MAE achieved on the real data was 41.17236328125.
```

exhibits somewhat less erratic behavior however the best performing model only achieves a 41.17 MAE which is as well the 1 convolution layer model. However this is also the only erratic model of this group.



```
By epoch 1000 The minimum MAE achieved by the training was 62.81932830810547.
By epoch 1000 The minimum MAE achieved on the real data was 66.97464752197266.
```



```
By epoch 1000 The minimum MAE achieved by the training was 31.458845138549805.
By epoch 1000 The minimum MAE achieved on the real data was 45.023773193359375.
number of seconds to run all experiments: 5918.22203040123
```

Hypothesis Conclusion

The training for these models is very erratic and has not stabilized in most cases the hypothesis has been proven to be false. MAE below the baseline was not reached. However, due to the erratic and unsettled training results more exploration and training is warranted.

Hypothesis 3:

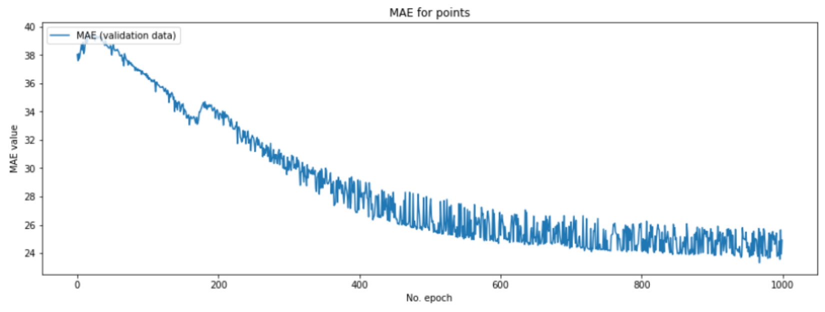
"Converting the input to grayscale and providing x and y axis value distribution information will lower MAE compared to the baseline model"

▼ Model 1

```
c_init = keras.initializers.Constant(.004)
b_init = keras.initializers.Constant(.004)

img_inputs = keras.Input(shape=(256, 256-64, 3))
x = img_inputs

# Simple grayscale

grayscale=tf.reduce_mean(x,axis=3,name = "grayscale")
y_mean = tf.reduce_mean(grayscale,axis=2,name = "y_mean")
x_mean = tf.reduce_mean(grayscale,axis=1,name = "x_mean")
x = layers.Flatten()(grayscale)
x = tf.concat([y_mean,x_mean,x],1)
x = layers.Dense(32, activation="relu",dtype="float32",kernel_initializer=c_init,bias_initializer=b_init)
(x)
x = layers.Dense(32, activation="relu",dtype="float32",kernel_initializer=c_init,bias_initializer=b_init)
(x)
x = layers.Dense(32, activation="relu",dtype="float32",kernel_initializer=c_init,bias_initializer=b_init)
(x)

outputs = layers.Dense(8,activation="relu",kernel_initializer=c_init,bias_initializer=b_init)(x)

model = keras.Model(inputs=img_inputs, outputs=outputs, name="FC_Model")
model.summary()
model.compile(loss="MAE",optimizer=tf.keras.optimizers.Adam(learning_rate=global_learning_rate))
```

Layers

- X and Y Mean



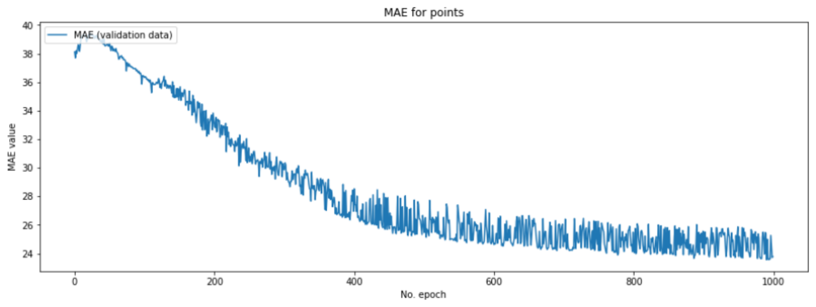
```
The minimum MAE achieved by the training was 7.628500461578369.
The minimum MAE achieved on the real data was 23.32073402404785.
```

▼ Model 2

```
class Gradient(tf.keras.layers.Layer):

    def __init__(self):
        super(Gradient, self).__init__()

    def call(self, inputs):
        return self.gradient(inputs)

    def gradient(self,a):
        rght = tf.concat((a[..., 1:], tf.expand_dims(a[..., -1], -1)), -1)
        left = tf.concat((tf.expand_dims(a[...,0], -1), a[..., :-1]), -1)
        ones = tf.ones_like(rght[..., 2:], tf.float32)
        one = tf.expand_dims(ones[...,0], -1)
        divi = tf.concat((one, ones*2, one), -1)
        return (rght-left) / divi

c_init = keras.initializers.Constant(.004)
b_init = keras.initializers.Constant(.004)

img_inputs = keras.Input(shape=(256, 256-64, 3))
x = img_inputs

# Simple grayscale
grayscale=tf.reduce_mean(x,axis=3,name = "grayscale")

# distribution information
y_mean = tf.reduce_mean(grayscale,axis=2,name = "y_mean")
x_mean = tf.reduce_mean(grayscale,axis=1,name = "x_mean")

y_grad = Gradient()(y_mean)
x_grad = Gradient()(x_mean)

x = layers.Flatten()(grayscale)
x = tf.concat([y_mean,x_mean,y_grad,x_grad,x],1)
x = layers.Dense(32, activation="relu",dtype="float32",kernel_initializer=c_init,bias_initializer=b_init)
(x)
x = layers.Dense(32, activation="relu",dtype="float32",kernel_initializer=c_init,bias_initializer=b_init)
(x)
x = layers.Dense(32, activation="relu",dtype="float32",kernel_initializer=c_init,bias_initializer=b_init)
(x)

outputs = layers.Dense(8,activation="relu",kernel_initializer=c_init,bias_initializer=b_init)(x)

model = keras.Model(inputs=img_inputs, outputs=outputs, name="FC_Model")
model.summary()
model.compile(loss="MAE",optimizer=tf.keras.optimizers.Adam(learning_rate=global_learning_rate))
```

Layers

- X and Y Mean
- Gradient of X and Y Mean



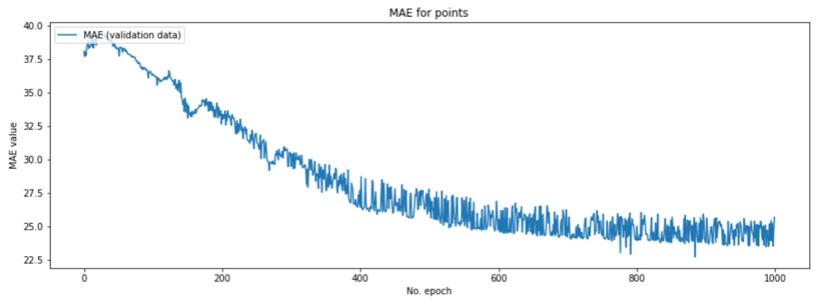
```
The minimum MAE achieved by the training was 7.646134853363037.
The minimum MAE achieved on the real data was 23.55926513671875.
```

▼ Model 3

```
class Gradient(tf.keras.layers.Layer):

    def __init__(self):
        super(Gradient, self).__init__()

    def call(self, inputs):
        return self.gradient(inputs)

    def gradient(self,a):
        rght = tf.concat((a[..., 1:], tf.expand_dims(a[..., -1], -1)), -1)
        left = tf.concat((tf.expand_dims(a[...,0], -1), a[..., :-1]), -1)
        ones = tf.ones_like(rght[..., 2:], tf.float32)
        one = tf.expand_dims(ones[...,0], -1)
        divi = tf.concat((one, ones*2, one), -1)
        return (rght-left) / divi


c_init = keras.initializers.Constant(.004)
b_init = keras.initializers.Constant(.004)

img_inputs = keras.Input(shape=(256, 256-64, 3))
x = img_inputs

# Simple grayscale
grayscale=tf.reduce_mean(x,axis=3,name = "grayscale")

# distribution information
y_mean = tf.reduce_mean(grayscale,axis=2,name = "y_mean")
x_mean = tf.reduce_mean(grayscale,axis=1,name = "x_mean")

y_grad = Gradient()(y_mean)
x_grad = Gradient()(x_mean)

y_grad_2 = Gradient()(y_grad)
x_grad_2 = Gradient()(x_grad)

x = layers.Flatten()(grayscale)
x = tf.concat([y_mean,x_mean,y_grad,x_grad,y_grad_2,x_grad_2,x],1)
x = layers.Dense(32, activation="relu",dtype="float32",kernel_initializer=c_init,bias_initializer=b_init)
(x)
x = layers.Dense(32, activation="relu",dtype="float32",kernel_initializer=c_init,bias_initializer=b_init)
(x)
x = layers.Dense(32, activation="relu",dtype="float32",kernel_initializer=c_init,bias_initializer=b_init)
(x)

outputs = layers.Dense(8,activation="relu",kernel_initializer=c_init,bias_initializer=b_init)(x)

model = keras.Model(inputs=img_inputs, outputs=outputs, name="FC_Model")
model.summary()
model.compile(loss="MAE",optimizer=tf.keras.optimizers.Adam(learning_rate=global_learning_rate))
```

Layers

- X and Y Mean
- Gradient of X and Y Mean
- Gradient of Gradient of X and Y Mean



```
The minimum MAE achieved by the training was 7.629327297210693.
The minimum MAE achieved on the real data was 22.734418869018555.
```

Hypothesis Conclusion

Providing x and y distribution information is shown not to significantly alter MAE performance. The hypothesis has been proven to be false.

Hypothesis 4:

Slicing the image into slices of 1/3 and only giving the model the corners will lower MAE of the baseline

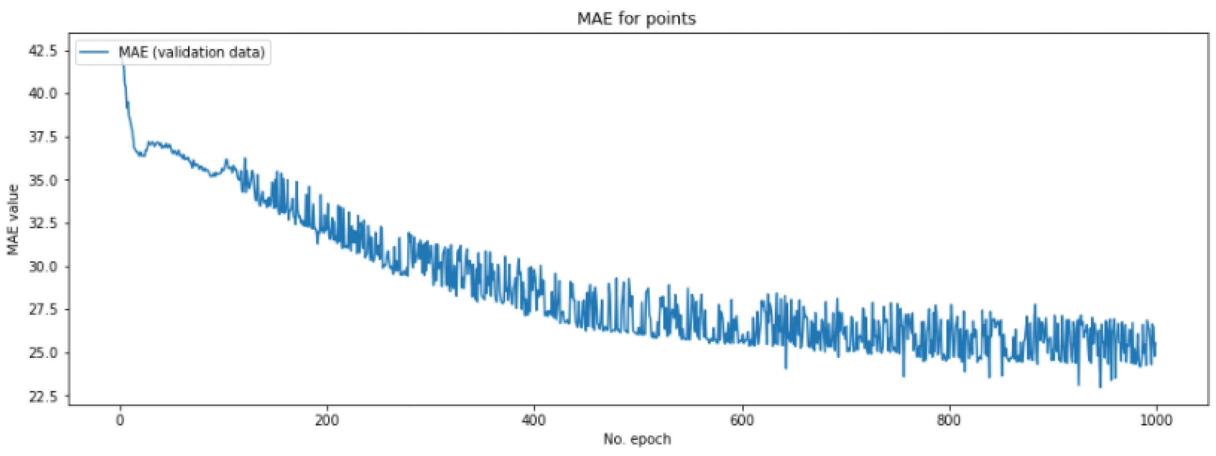
▼ Custom Layer

```
class GetCorners(tf.keras.layers.Layer):

    def __init__(self,divisions:int=3):
        super(GetCorners, self).__init__()
        self.divisions=divisions

    def call(self, inputs):
        h = inputs.shape[1]
        w = inputs.shape[2]
        h_seg = int(h/self.divisions)
        w_seg = int(w/self.divisions)
        h_mid = int(h - h_seg)
        w_mid = int(w - w_seg)

        first = (h_seg,w_seg)
        middle = (h_mid,w_mid)

        corner1 = inputs[...,:first[0] ,:first[1]   ,:]
        corner2 = inputs[...,middle[0]:,:first[1]  ,:]
        corner3 = inputs[...,middle[0]:,middle[1]: ,:]
        corner4 = inputs[...,:first[0], middle[1]: ,:]
        out1 = tf.concat([corner1,corner2],1)
        out2 = tf.concat([corner4,corner3],1)
        out3 = tf.concat([out1,out2],2)
        return out3
```

▼ Modal A: Sliced by thirds corners.

```
c_init = keras.initializers.Constant(.004)
b_init = keras.initializers.Constant(.004)

img_inputs = keras.Input(shape=(256, 256-64, 3))
x = img_inputs

# Simple grayscale

corners=GetCorners(divisions=3)(x)

x = layers.Flatten()(corners)

x = layers.Dense(32, activation="relu",dtype="float32",kernel_initializer=c_init,bias_initializer=b_init)(x)
x = layers.Dense(32, activation="relu",dtype="float32",kernel_initializer=c_init,bias_initializer=b_init)(x)
x = layers.Dense(32, activation="relu",dtype="float32",kernel_initializer=c_init,bias_initializer=b_init)(x)

outputs = layers.Dense(8,activation="relu",kernel_initializer=c_init,bias_initializer=b_init)(x)
```

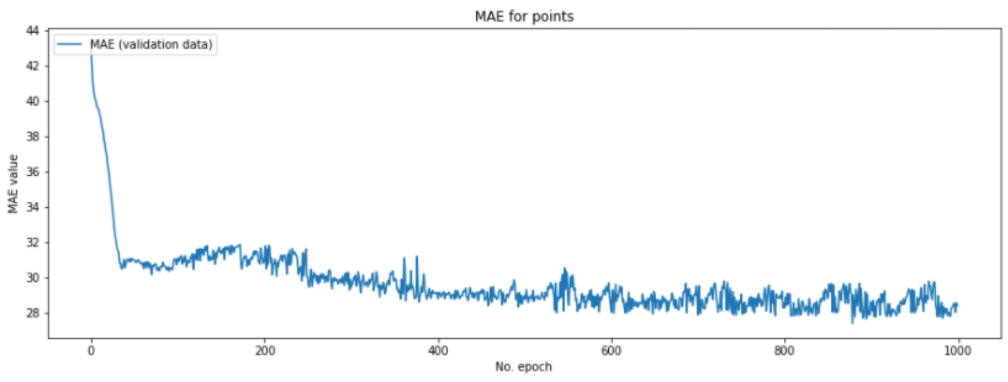
```
model = keras.Model(inputs=img_inputs, outputs=outputs, name="FC_Model")
model.summary()
model.compile(loss="MAE",optimizer=tf.keras.optimizers.Adam(learning_rate=global_learning_rate))
```



```
The minimum MAE achieved by the training was 7.6425933837890625.
The minimum MAE achieved on the real data was 22.97004508972168.
runtime: 641.1639695167542 seconds
```

▼ Model B: Sliced By fourths corners

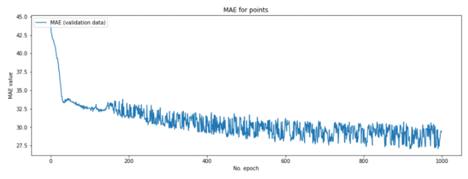
```
c_init = keras.initializers.Constant(.004)
b_init = keras.initializers.Constant(.004)

img_inputs = keras.Input(shape=(256, 256-64, 3))
x = img_inputs

# Simple grayscale

corners=GetCorners(divisions=4)(x)

x = layers.Flatten()(corners)

x = layers.Dense(32, activation="relu",dtype="float32",kernel_initializer=c_init,bias_initializer=b_init)
(x)
x = layers.Dense(32, activation="relu",dtype="float32",kernel_initializer=c_init,bias_initializer=b_init)
(x)
x = layers.Dense(32, activation="relu",dtype="float32",kernel_initializer=c_init,bias_initializer=b_init)
(x)

outputs = layers.Dense(8,activation="relu",kernel_initializer=c_init,bias_initializer=b_init)(x)

model = keras.Model(inputs=img_inputs, outputs=outputs, name="FC_Model")
model.summary()
model.compile(loss="MAE",optimizer=tf.keras.optimizers.Adam(learning_rate=global_learning_rate))
```



```
The minimum MAE achieved by the training was 7.730838775634766.
The minimum MAE achieved on the real data was 27.369775772094727.
runtime: 793.3310775756836 seconds
```

▼ Modal C: Sliced by fifths corners

```
c_init = keras.initializers.Constant(.004)
b_init = keras.initializers.Constant(.004)

img_inputs = keras.Input(shape=(256, 256-64, 3))
x = img_inputs

# Simple grayscale

corners=GetCorners(divisions=5)(x)

x = layers.Flatten()(corners)

x = layers.Dense(32, activation="relu",dtype="float32",kernel_initializer=c_init,bias_initializer=b_init)(x)
x = layers.Dense(32, activation="relu",dtype="float32",kernel_initializer=c_init,bias_initializer=b_init)(x)
x = layers.Dense(32, activation="relu",dtype="float32",kernel_initializer=c_init,bias_initializer=b_init)(x)

outputs = layers.Dense(8,activation="relu",kernel_initializer=c_init,bias_initializer=b_init)(x)

model = keras.Model(inputs=img_inputs, outputs=outputs, name="FC_Model")
model.summary()
model.compile(loss="MAE",optimizer=tf.keras.optimizers.Adam(learning_rate=global_learning_rate))
```



```
The minimum MAE achieved by the training was 7.645534038543701.
The minimum MAE achieved on the real data was 27.062637329101562.
runtime: 671.1397631168365 seconds
```

Hypothesis Conclusion

These sliced corner models perform very similar to the baseline model, however the hypothesis has been proven false. Slicing does not improve MAE.

Reflection

Is it possible to design a model infer user input in story_photo_transformer.py. And will the result be of comparable image quality? Is it possible using the synthetic dataset provided by generate.py and samples of user input provided by story_photo_transformer.py to generate coordinates with less than 8pix mean average error?

My experiments here reflect on only part of the research question, they show that it is possible to replace the user input in story_photo_transformer.py. And that using synthetic data does improve the predictions of the model. The best outcome here would of been to find a model that trained significantly better than the baseline, no such model has become apparent when we look at MAE reached per 1000 epochs. However MAE/Epoch is not the true value proposition. It is important to understand that the true value of a model is the minimum MAE that it will achieved given an amount of training and data that is reasonably achievable by the organization.

With this understanding in mind several features of these models become more important:

- The convolutional models showed erratic training, this could mean that they are exploring a larger solution space and may indeed end up with the lowest MAE with more training
- The grayscale models with distribution/gradient information have less than 50% of the parameters to tune then that of the baseline. These models train much quicker and show similar MAE performance.
- The sliced models also have many less parameters to be trained and offer similar MAE performance

Recommendation

Further experimentation is warranted, however it is also important to understand that it is possible that all of these models are sufficient to infer user input if they are also trained on a sufficient number of real user data samples.

Our next set of experiments should include training on whatever limited user data is available.