Name:Matthew Seman

In [1]:
```
%matplotlib inline
import cv2
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import style
style.use('ggplot')
from IPython import display
import pandas as pd
import tensorflow as tf
```

In [2]:
```
## helper funciton for loading the horses data
#make sure the horses data is in the same directory as the notebook
#do not change this function
def load_horses_orig(path, image_size):
    mask_path = path + 'masks/'
    image_path = path + 'images/'
    images = []
    masks = []
    test_images= []
    test_masks =[]
    for i in range(328):

        orig_im = cv2.imread(image_path + 'image-{}.png'.format(i))
        orig_im= cv2.cvtColor(orig_im, cv2.COLOR_RGB2BGR)

        low_im = cv2.resize(orig_im, dsize=(image_size, image_size))

        orig_mask = cv2.imread(mask_path + 'mask-{}.png'.format(i))
        low_mask = cv2.resize(orig_mask, dsize=(image_size, image_size))
        low_mask = cv2.cvtColor(low_mask, cv2.COLOR_RGB2GRAY)
        bin_mask = (low_mask > 0) + 0


        images.append(low_im)
        masks.append(bin_mask)


    xtest = np.reshape(np.array(images[250:]), (-1,image_size*image_size*3))
    ytest = np.reshape(np.array(masks[250:]), (-1, image_size * image_size))
    xdata = np.reshape(np.array(images[:200]), (-1,image_size*image_size*3))
    ydata = np.reshape(np.array(masks[:200]), (-1, image_size * image_size))
    yval =  np.reshape(np.array(masks[200:250]), (-1, image_size * image_size))
    xval = np.reshape(np.array(images[200:250]), (-1,image_size*image_size*3))


    return xdata, xval, xtest, ydata, yval, ytest
```

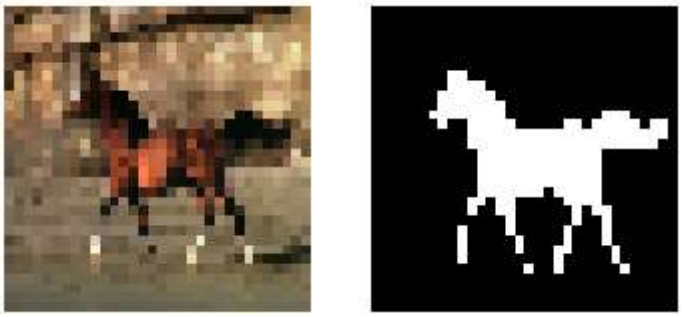
In [3]:
```
#change the path address if put the data somewhere else
path = './horses/'
image_size = 32;
xdata, xval, xtest, ydata, yval, ytest = load_horses_orig(path, image_size)
```

In [4]:

```
#helper function to drawing horse and its mask
def draw(image, mask):
    fig, (ax1,ax2) = plt.subplots(1,2)
    ax1.axis('off')
    ax2.axis('off')
    ax1.imshow(np.reshape(image, (image_size,image_size,3)))
    ax2.imshow(np.reshape(mask, (image_size,image_size)), cmap=plt.cm.gray)
    plt.show()
```

The task is to predict the mask for the horse given the image Mask is binary image shows the presence of the horse.

In [5]:
```
draw(xdata[0], ydata[0])
```



In [6]:
```
train_size = xdata.shape[0]
batch_size = 10
train_dataset = (tf.data.Dataset.from_tensor_slices(np.hstack((xdata, ydata)))
                 .shuffle(train_size).batch(batch_size))
```

The intersection over union (IOU) is a metric for measuring the performance of image segmentation. The perfect segmentation receives IOU of one.

In [7]:
```
# Do not change this cell
def iou(ytrue, yprediction):
    yp = yprediction
    yt = ytrue
    yp = yp > 0.5 + 0
    intersect = np.sum(np.minimum(yp, yt),1)
    union = np.sum(np.maximum(yp, yt),1)
    return np.average(intersect / (union+0.0))
```

In [8]:
```
assert iou(ydata, ydata) == 1.0
```

We can use feedforwad MLP or CNN model for image segmentation. Here the input is the image and the output is the segmentation mask.

In [9]:
```
class NeuralNet(tf.keras.Model):

    def construct_CNN(self):
        # COMPLETE ME
        self.model = tf.keras.Sequential(
```

```
            [
                tf.keras.layers.Conv2D(32, kernel_size=3, activation='relu', padding='s
                tf.keras.layers.BatchNormalization(),
                tf.keras.layers.Conv2D(32, kernel_size=3, activation='relu', padding='s
                tf.keras.layers.BatchNormalization(),
                tf.keras.layers.Conv2D(32, kernel_size=3, activation='relu', padding='s
                tf.keras.layers.BatchNormalization(),
                tf.keras.layers.MaxPooling2D((2,2)),
                tf.keras.layers.Dense(16, activation='relu'),
                tf.keras.layers.Dense(16, activation='relu'),
                tf.keras.layers.UpSampling2D((2,2)),
                tf.keras.layers.Conv2DTranspose(32, kernel_size=3, activation='relu', p
                tf.keras.layers.BatchNormalization(),
                tf.keras.layers.Conv2DTranspose(32, kernel_size=3, activation='relu', p
                tf.keras.layers.BatchNormalization(),
                tf.keras.layers.Conv2DTranspose(32, kernel_size=3, activation='relu', p
                tf.keras.layers.BatchNormalization(),
                tf.keras.layers.Flatten(),
                tf.keras.layers.Dense(128, activation='relu'),
                tf.keras.layers.Dropout(.2),
                tf.keras.layers.Dense(image_size*image_size)

            ]
        )
        pass


    def construct_MLP(self):
         self.model = tf.keras.Sequential(
            [
                tf.keras.layers.InputLayer(input_shape=(image_size, image_size, 3), dty
                tf.keras.layers.Flatten(),
                tf.keras.layers.Dense(500, activation='relu'),
                tf.keras.layers.Dropout(.2, input_shape=(500,)),
                tf.keras.layers.Dense(20, activation='relu'),
                tf.keras.layers.Dense(image_size*image_size)
            ]
         )


    def __init__(self, network_type = 'MLP'):
        super(NeuralNet, self).__init__()
        #DOCUMENT ME
        #SGD is a stochastic gradient descent optimizer that also allows for use of mom
        #momentum is a smoothing hyperparameter to increase descents and dampen oscilla

        #self.optimizer = tf.keras.optimizers.SGD(0.1)

        #What are the other available optimizers

        #Reduced Learning rate on Adam optimizer gives best performance with current CN
        self.optimizer = tf.keras.optimizers.Adam(0.001)

        #Test your code with different optimizers




        if network_type == 'MLP':
            self.construct_MLP();
```

```
        else:
            self.construct_CNN();


    #DOCUMENT ME
    #Defines the forward prediction for the model in which the sigmoid of x is taken us
    def forward(self, x, predict=True):
        if predict:
            return tf.nn.sigmoid(self.model(x))
        return self.model(x)

    def get_loss(self, x, yt):
        x = tf.reshape(x, (-1, image_size, image_size, 3))
        yt = tf.reshape(yt, (-1, image_size * image_size))
        ylogits = self.forward(x, predict=False);

        #DOCUMENT ME
        #What is the difference of sigmoid_cross_entropy_with_logits and sigmoid_cross_
        #sigmoid cross entropy with logits uses logits(log-odds function) and labels of
        cross_ent = tf.nn.sigmoid_cross_entropy_with_logits(logits=ylogits, labels=yt);


        return tf.reduce_sum(cross_ent)


    def train_step(self, xbatch,ybatch):

        #DOCUMENT ME
        #GradientTape() watches variables within its context, in this case loss, and re
        #Used for automatic differentiation
        with tf.GradientTape() as tape:
            loss = self.get_loss(xbatch, ybatch)
        #DOCUMENT ME
        #Calculate the gradient of the target variable from the GradientTape, in this c
        gradients = tape.gradient(loss, model.trainable_variables)

        #DOCUMENT ME
        #apply_gradients applies gradient descent and updates the weights of the model
        self.optimizer.apply_gradients(zip(gradients, model.trainable_variables))
        return tf.reduce_sum(loss)

    def predict(self, x):
        y = self.forward(x, predict=True)
        return y.numpy() #tf.reshape(y, (-1, image_size, image_size)).numpy()
```

In [10]:

```
model = NeuralNet('CNN')
#model = NeuralNet('MLP')
train_iou = []
val_iou = []
test_iou = []
epoch = 1;
best_val_iou = -1;
best_test_iou = -1
```

In [11]:

```
#Check Model
modelCheck = tf.keras.Sequential(
            [
```

```
                tf.keras.layers.Conv2D(32, kernel_size=3, activation='relu', padding='s
                tf.keras.layers.BatchNormalization(),
                tf.keras.layers.Conv2D(32, kernel_size=3, activation='relu', padding='s
                tf.keras.layers.BatchNormalization(),
                tf.keras.layers.Conv2D(32, kernel_size=3, activation='relu', padding='s
                tf.keras.layers.BatchNormalization(),
                tf.keras.layers.MaxPooling2D((2,2)),
                tf.keras.layers.Dense(16, activation='relu'),
                tf.keras.layers.Dense(16, activation='relu'),
                tf.keras.layers.UpSampling2D((2,2)),
                tf.keras.layers.Conv2DTranspose(32, kernel_size=3, activation='relu', p
                tf.keras.layers.BatchNormalization(),
                tf.keras.layers.Conv2DTranspose(32, kernel_size=3, activation='relu', p
                tf.keras.layers.BatchNormalization(),
                tf.keras.layers.Conv2DTranspose(32, kernel_size=3, activation='relu', p
                tf.keras.layers.BatchNormalization(),
                tf.keras.layers.Flatten(),
                tf.keras.layers.Dense(128, activation='relu'),
                tf.keras.layers.Dropout(.2),
                tf.keras.layers.Dense(image_size*image_size)
            ]
        )
modelCheck.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 32, 32, 32)        896

 batch_normalization_6 (Batc  (None, 32, 32, 32)       128
 hNormalization)

 conv2d_4 (Conv2D)           (None, 32, 32, 32)        9248

 batch_normalization_7 (Batc  (None, 32, 32, 32)       128
 hNormalization)

 conv2d_5 (Conv2D)           (None, 32, 32, 32)        9248

 batch_normalization_8 (Batc  (None, 32, 32, 32)       128
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 dense_4 (Dense)             (None, 16, 16, 16)        528

 dense_5 (Dense)             (None, 16, 16, 16)        272

 up_sampling2d_1 (UpSampling  (None, 32, 32, 16)       0
 2D)

 conv2d_transpose_3 (Conv2DT  (None, 32, 32, 32)       4640
 ranspose)

 batch_normalization_9 (Batc  (None, 32, 32, 32)       128
 hNormalization)

 conv2d_transpose_4 (Conv2DT  (None, 32, 32, 32)       9248
```

```
ranspose)

batch_normalization_10 (Bat  (None, 32, 32, 32)       128
chNormalization)

conv2d_transpose_5 (Conv2DT  (None, 32, 32, 32)       9248
ranspose)

batch_normalization_11 (Bat  (None, 32, 32, 32)       128
chNormalization)

flatten_1 (Flatten)          (None, 32768)            0

dense_6 (Dense)              (None, 128)              4194432

dropout_1 (Dropout)          (None, 128)              0

dense_7 (Dense)              (None, 1024)             132096

=================================================================
Total params: 4,370,624
Trainable params: 4,370,240
Non-trainable params: 384
_________________________________________________________________
```

In [12]:

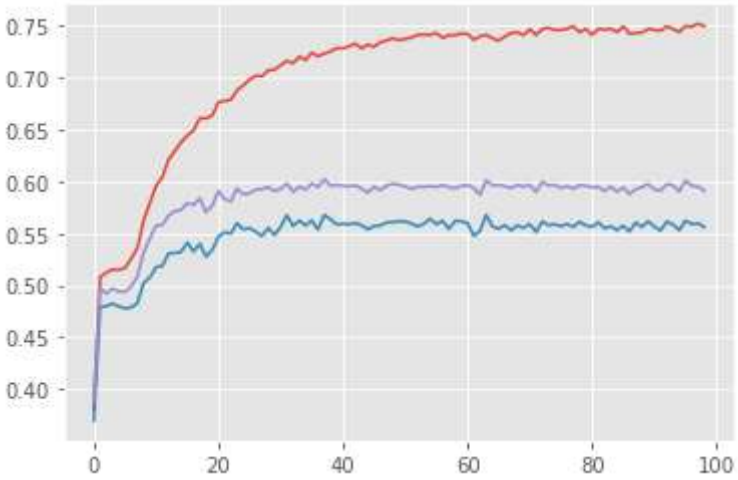
```
#Adopt similar training loop for other problems

#Originally 5000
max_epoch = 100

while epoch < max_epoch:
    loss = 0.0
    for batch in train_dataset:
        xbatch = batch[:,:image_size*image_size*3]
        ybatch = batch[:,image_size*image_size*3:]
        xbatch = tf.cast(xbatch, tf.float32);
        ybatch = tf.cast(ybatch, tf.float32);
        loss += model.train_step(xbatch, ybatch);

    ydata_pred = model.predict(tf.reshape(xdata, (-1, image_size, image_size, 3 )));
    yval_pred = model.predict(tf.reshape(xval, (-1, image_size, image_size, 3 )));
    ytest_pred = model.predict(tf.reshape(xtest, (-1, image_size, image_size, 3 )));

    train_iou.append(iou(ydata,ydata_pred))
    val_iou.append(iou(yval,yval_pred))
    test_iou.append(iou(ytest,ytest_pred))

    if val_iou[-1] > best_val_iou:
        best_test_iou = test_iou[-1]
        best_val_iou = val_iou[-1]
        model.save_weights("best_model");


    print("Epoch {}, Loss: {:0.3}, IOU - Train: {:0.3} Valid: {:0.3} Test: {:0.3}".form



    epoch += 1
```

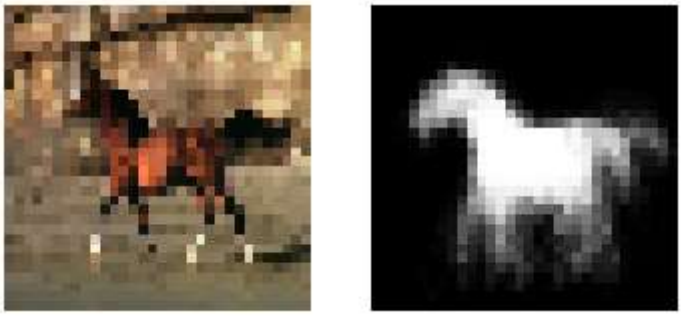
```
Epoch 1, Loss: 1.39e+05, IOU - Train: 0.38 Valid: 0.37 Test: 0.387
Epoch 2, Loss: 8.7e+04, IOU - Train: 0.508 Valid: 0.479 Test: 0.497
Epoch 3, Loss: 7.19e+04, IOU - Train: 0.512 Valid: 0.48 Test: 0.492
Epoch 4, Loss: 7.04e+04, IOU - Train: 0.515 Valid: 0.482 Test: 0.497
Epoch 5, Loss: 6.88e+04, IOU - Train: 0.515 Valid: 0.48 Test: 0.497
Epoch 6, Loss: 6.84e+04, IOU - Train: 0.517 Valid: 0.478 Test: 0.497
Epoch 7, Loss: 6.74e+04, IOU - Train: 0.526 Valid: 0.479 Test: 0.497
Epoch 8, Loss: 6.59e+04, IOU - Train: 0.536 Valid: 0.483 Test: 0.509
Epoch 9, Loss: 6.41e+04, IOU - Train: 0.563 Valid: 0.503 Test: 0.532
Epoch 10, Loss: 6.2e+04, IOU - Train: 0.58 Valid: 0.508 Test: 0.545
Epoch 11, Loss: 5.99e+04, IOU - Train: 0.596 Valid: 0.518 Test: 0.558
Epoch 12, Loss: 5.67e+04, IOU - Train: 0.604 Valid: 0.519 Test: 0.558
Epoch 13, Loss: 5.43e+04, IOU - Train: 0.621 Valid: 0.531 Test: 0.567
Epoch 14, Loss: 5.25e+04, IOU - Train: 0.63 Valid: 0.531 Test: 0.572
Epoch 15, Loss: 5.07e+04, IOU - Train: 0.638 Valid: 0.532 Test: 0.573
Epoch 16, Loss: 4.92e+04, IOU - Train: 0.644 Valid: 0.541 Test: 0.579
Epoch 17, Loss: 4.83e+04, IOU - Train: 0.649 Valid: 0.532 Test: 0.579
Epoch 18, Loss: 4.74e+04, IOU - Train: 0.661 Valid: 0.54 Test: 0.579
Epoch 19, Loss: 4.63e+04, IOU - Train: 0.661 Valid: 0.528 Test: 0.579
Epoch 20, Loss: 4.55e+04, IOU - Train: 0.663 Valid: 0.534 Test: 0.579
Epoch 21, Loss: 4.48e+04, IOU - Train: 0.676 Valid: 0.547 Test: 0.591
Epoch 22, Loss: 4.41e+04, IOU - Train: 0.678 Valid: 0.551 Test: 0.583
Epoch 23, Loss: 4.37e+04, IOU - Train: 0.679 Valid: 0.55 Test: 0.583
Epoch 24, Loss: 4.29e+04, IOU - Train: 0.688 Valid: 0.56 Test: 0.594
Epoch 25, Loss: 4.21e+04, IOU - Train: 0.693 Valid: 0.554 Test: 0.594
Epoch 26, Loss: 4.14e+04, IOU - Train: 0.698 Valid: 0.555 Test: 0.594
Epoch 27, Loss: 4.06e+04, IOU - Train: 0.702 Valid: 0.552 Test: 0.594
Epoch 28, Loss: 4.05e+04, IOU - Train: 0.702 Valid: 0.548 Test: 0.594
Epoch 29, Loss: 3.98e+04, IOU - Train: 0.707 Valid: 0.555 Test: 0.594
Epoch 30, Loss: 3.92e+04, IOU - Train: 0.708 Valid: 0.548 Test: 0.594
Epoch 31, Loss: 3.85e+04, IOU - Train: 0.712 Valid: 0.556 Test: 0.594
Epoch 32, Loss: 3.82e+04, IOU - Train: 0.716 Valid: 0.568 Test: 0.598
Epoch 33, Loss: 3.8e+04, IOU - Train: 0.714 Valid: 0.557 Test: 0.598
Epoch 34, Loss: 3.79e+04, IOU - Train: 0.72 Valid: 0.562 Test: 0.598
Epoch 35, Loss: 3.77e+04, IOU - Train: 0.717 Valid: 0.558 Test: 0.598
Epoch 36, Loss: 3.74e+04, IOU - Train: 0.724 Valid: 0.563 Test: 0.598
Epoch 37, Loss: 3.71e+04, IOU - Train: 0.721 Valid: 0.554 Test: 0.598
Epoch 38, Loss: 3.7e+04, IOU - Train: 0.723 Valid: 0.568 Test: 0.602
Epoch 39, Loss: 3.73e+04, IOU - Train: 0.725 Valid: 0.564 Test: 0.602
Epoch 40, Loss: 3.66e+04, IOU - Train: 0.729 Valid: 0.559 Test: 0.602
Epoch 41, Loss: 3.61e+04, IOU - Train: 0.728 Valid: 0.559 Test: 0.602
Epoch 42, Loss: 3.58e+04, IOU - Train: 0.73 Valid: 0.559 Test: 0.602
Epoch 43, Loss: 3.56e+04, IOU - Train: 0.733 Valid: 0.56 Test: 0.602
Epoch 44, Loss: 3.57e+04, IOU - Train: 0.728 Valid: 0.558 Test: 0.602
Epoch 45, Loss: 3.56e+04, IOU - Train: 0.732 Valid: 0.554 Test: 0.602
Epoch 46, Loss: 3.56e+04, IOU - Train: 0.73 Valid: 0.557 Test: 0.602
Epoch 47, Loss: 3.55e+04, IOU - Train: 0.734 Valid: 0.557 Test: 0.602
Epoch 48, Loss: 3.52e+04, IOU - Train: 0.736 Valid: 0.561 Test: 0.602
Epoch 49, Loss: 3.54e+04, IOU - Train: 0.738 Valid: 0.561 Test: 0.602
Epoch 50, Loss: 3.52e+04, IOU - Train: 0.736 Valid: 0.562 Test: 0.602
Epoch 51, Loss: 3.5e+04, IOU - Train: 0.738 Valid: 0.562 Test: 0.602
Epoch 52, Loss: 3.5e+04, IOU - Train: 0.739 Valid: 0.559 Test: 0.602
Epoch 53, Loss: 3.48e+04, IOU - Train: 0.741 Valid: 0.557 Test: 0.602
Epoch 54, Loss: 3.43e+04, IOU - Train: 0.742 Valid: 0.559 Test: 0.602
Epoch 55, Loss: 3.44e+04, IOU - Train: 0.741 Valid: 0.564 Test: 0.602
Epoch 56, Loss: 3.42e+04, IOU - Train: 0.743 Valid: 0.559 Test: 0.602
Epoch 57, Loss: 3.42e+04, IOU - Train: 0.739 Valid: 0.562 Test: 0.602
Epoch 58, Loss: 3.43e+04, IOU - Train: 0.741 Valid: 0.554 Test: 0.602
Epoch 59, Loss: 3.45e+04, IOU - Train: 0.741 Valid: 0.562 Test: 0.602
Epoch 60, Loss: 3.41e+04, IOU - Train: 0.743 Valid: 0.562 Test: 0.602
```

```
Epoch 61, Loss: 3.44e+04, IOU - Train: 0.742 Valid: 0.56 Test: 0.602
Epoch 62, Loss: 3.44e+04, IOU - Train: 0.737 Valid: 0.548 Test: 0.602
Epoch 63, Loss: 3.42e+04, IOU - Train: 0.74 Valid: 0.553 Test: 0.602
Epoch 64, Loss: 3.43e+04, IOU - Train: 0.741 Valid: 0.568 Test: 0.601
Epoch 65, Loss: 3.45e+04, IOU - Train: 0.738 Valid: 0.557 Test: 0.601
Epoch 66, Loss: 3.44e+04, IOU - Train: 0.736 Valid: 0.555 Test: 0.601
Epoch 67, Loss: 3.44e+04, IOU - Train: 0.74 Valid: 0.558 Test: 0.601
Epoch 68, Loss: 3.41e+04, IOU - Train: 0.743 Valid: 0.553 Test: 0.601
Epoch 69, Loss: 3.39e+04, IOU - Train: 0.744 Valid: 0.558 Test: 0.601
Epoch 70, Loss: 3.38e+04, IOU - Train: 0.741 Valid: 0.555 Test: 0.601
Epoch 71, Loss: 3.36e+04, IOU - Train: 0.747 Valid: 0.559 Test: 0.601
Epoch 72, Loss: 3.37e+04, IOU - Train: 0.741 Valid: 0.551 Test: 0.601
Epoch 73, Loss: 3.35e+04, IOU - Train: 0.747 Valid: 0.562 Test: 0.601
Epoch 74, Loss: 3.35e+04, IOU - Train: 0.748 Valid: 0.558 Test: 0.601
Epoch 75, Loss: 3.33e+04, IOU - Train: 0.746 Valid: 0.559 Test: 0.601
Epoch 76, Loss: 3.32e+04, IOU - Train: 0.746 Valid: 0.557 Test: 0.601
Epoch 77, Loss: 3.31e+04, IOU - Train: 0.747 Valid: 0.559 Test: 0.601
Epoch 78, Loss: 3.31e+04, IOU - Train: 0.75 Valid: 0.556 Test: 0.601
Epoch 79, Loss: 3.31e+04, IOU - Train: 0.744 Valid: 0.561 Test: 0.601
Epoch 80, Loss: 3.33e+04, IOU - Train: 0.747 Valid: 0.558 Test: 0.601
Epoch 81, Loss: 3.37e+04, IOU - Train: 0.741 Valid: 0.556 Test: 0.601
Epoch 82, Loss: 3.35e+04, IOU - Train: 0.747 Valid: 0.561 Test: 0.601
Epoch 83, Loss: 3.32e+04, IOU - Train: 0.747 Valid: 0.555 Test: 0.601
Epoch 84, Loss: 3.32e+04, IOU - Train: 0.747 Valid: 0.557 Test: 0.601
Epoch 85, Loss: 3.33e+04, IOU - Train: 0.744 Valid: 0.553 Test: 0.601
Epoch 86, Loss: 3.32e+04, IOU - Train: 0.75 Valid: 0.557 Test: 0.601
Epoch 87, Loss: 3.31e+04, IOU - Train: 0.743 Valid: 0.552 Test: 0.601
Epoch 88, Loss: 3.33e+04, IOU - Train: 0.743 Valid: 0.561 Test: 0.601
Epoch 89, Loss: 3.34e+04, IOU - Train: 0.744 Valid: 0.556 Test: 0.601
Epoch 90, Loss: 3.35e+04, IOU - Train: 0.747 Valid: 0.562 Test: 0.601
Epoch 91, Loss: 3.32e+04, IOU - Train: 0.746 Valid: 0.557 Test: 0.601
Epoch 92, Loss: 3.29e+04, IOU - Train: 0.746 Valid: 0.553 Test: 0.601
Epoch 93, Loss: 3.31e+04, IOU - Train: 0.75 Valid: 0.561 Test: 0.601
Epoch 94, Loss: 3.32e+04, IOU - Train: 0.747 Valid: 0.558 Test: 0.601
Epoch 95, Loss: 3.31e+04, IOU - Train: 0.744 Valid: 0.553 Test: 0.601
Epoch 96, Loss: 3.3e+04, IOU - Train: 0.75 Valid: 0.562 Test: 0.601
Epoch 97, Loss: 3.28e+04, IOU - Train: 0.749 Valid: 0.559 Test: 0.601
Epoch 98, Loss: 3.28e+04, IOU - Train: 0.752 Valid: 0.56 Test: 0.601
Epoch 99, Loss: 3.27e+04, IOU - Train: 0.75 Valid: 0.556 Test: 0.601
```

In [13]:

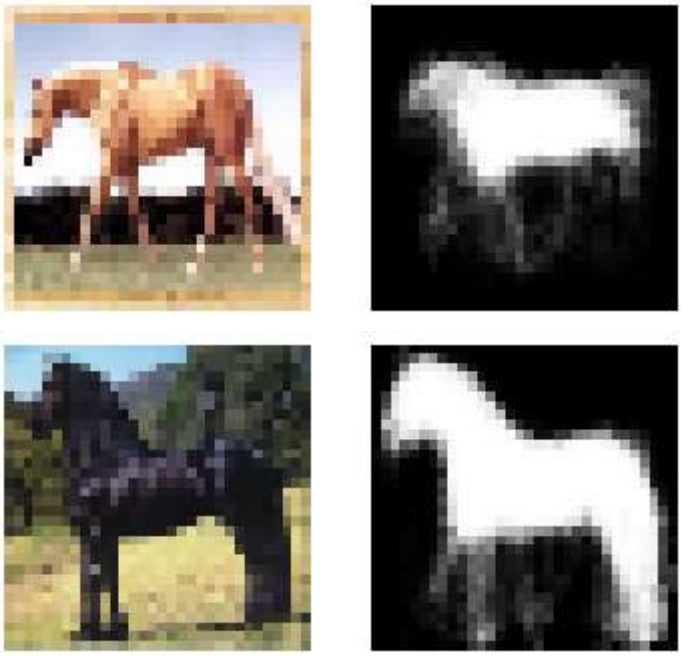
```
fig, ax = plt.subplots(1,1)
pd.Series(train_iou).plot(ax=ax)
pd.Series(val_iou).plot(ax=ax)
pd.Series(test_iou).plot(ax=ax)
fig.canvas.draw()

draw(xdata[0], ydata_pred[0])
```

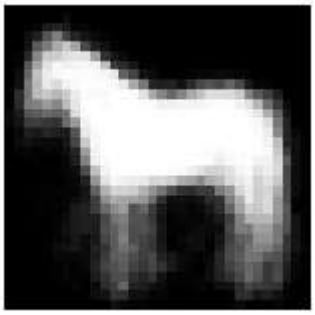




In [14]:
```
model.load_weights("best_model")
ypred = model.predict(tf.reshape(xtest, (-1, image_size, image_size, 3 )));
```

In [15]:
```
draw(xtest[10], ypred[10])
draw(xtest[25], ypred[25])
draw(xtest[20], ypred[20])
```





In [16]:
```
iou(ytest, ypred)
```

Out[16]:
```
0.6013728945710276
```

In [ ]:
```

```