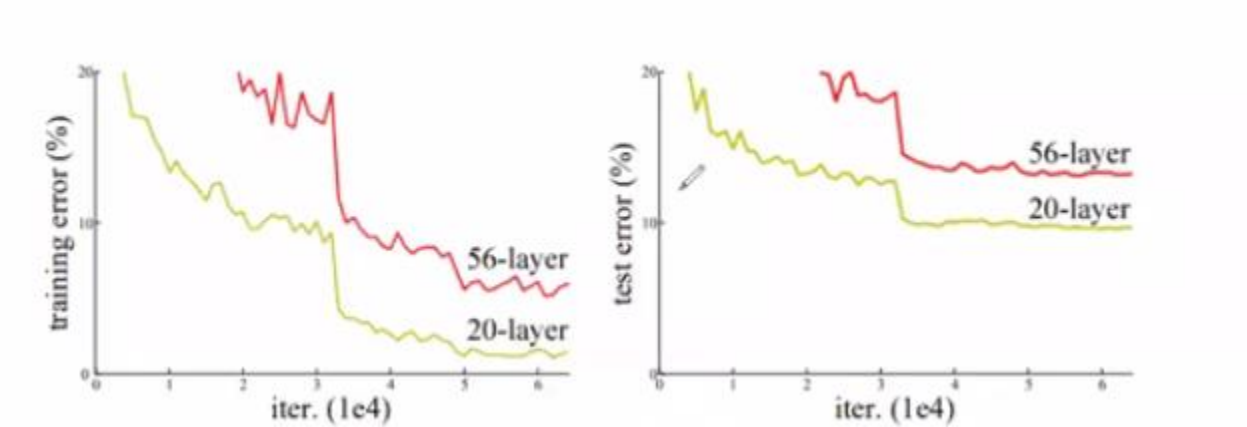
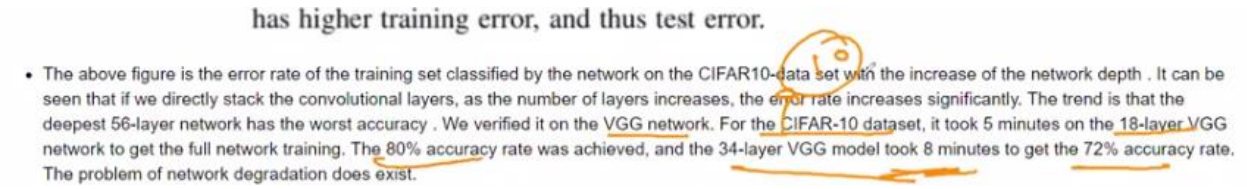
Resnet



If the depth of the network is increasing, after certain iterations you will be able to find out 20 layer network performs in terms of an error much better than 56 layer.

As we discussed VGG16 and VGG19 are pretty good as compare to LeNet or compare to AlexNet. In VGG16 instead of using different kinds of kernels they have taken a pair of kernels which means 3X3 3 convolution... 3X3 4 convolution and so on. Here we have fixed the kernel size.

In VGG16 we know the advantage like extracting the feature out of the feature. In that case if I have a very high dimensional I will to unwrap most of the properties that was the idea behind the deep network. There is better accuracy for 16 and 19 layer network. Suppose beyond the layer 20 they have observed that idea is going to fail. In each iteration it should increase the accuracy but it has started decreasing and there was a various reasons behind this particular networks.


has higher training error, and thus test error.

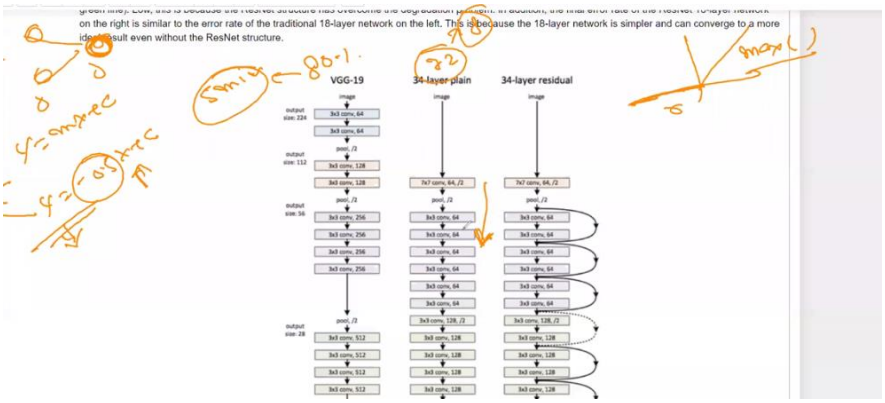
- The above figure is the error rate of the training set classified by the network on the CIFAR10-data set with the increase of the network depth . It can be seen that if we directly stack the convolutional layers, as the number of layers increases, the error rate increases significantly. The trend is that the deepest 56-layer network has the worst accuracy . We verified it on the VGG network. For the CIFAR-10 dataset, it took 5 minutes on the 18-layer VGG network to get the full network training. The 80% accuracy rate was achieved, and the 34-layer VGG model took 8 minutes to get the 72% accuracy rate. The problem of network degradation does exist.

So people from **Microsoft** have created a new kind of idea or Network called as **Residual Network (ResNet)** to handle this huge kind of networks with better accuracy.

Y=mx+c if m is negative like -0.5*x + c in this case x increase y will decrease.

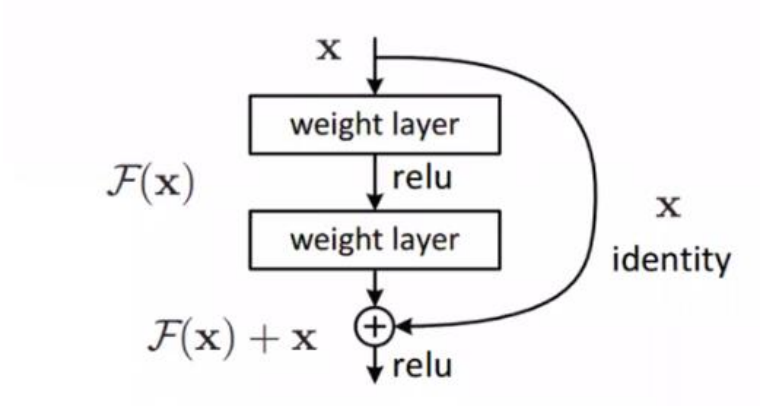
Ina neural network whenever we try to send the datasets inside the activation function. It is nothing but the summation of (summation of wx + b) weights. If there is negative weight unnecessarily I'm trying to train it from the negative to the positive and I have supposed to use lots of parameter tuning and trying to train the weights. May be I will not able to find out the better weight.

Weight training problem – in a back propagation so, if you will start multiplying the weights for sure the next weight will be depends upon the previous weight and for sure there will be effect. Convergence wise it will be not able to converge properly, even if am going to use Adam optimizer, Ada delta … we know there is a limitation it will not able to converge rapidly or absolute minima. This kind of issue occur in this kind of network 34 layer

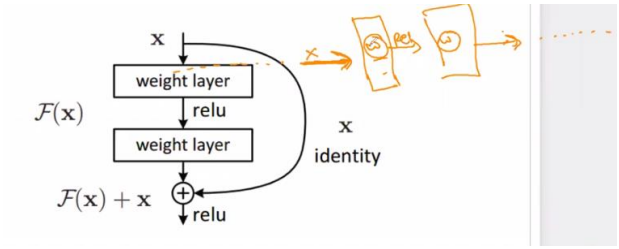


They have started to thinking about the new idea.

Residual Network Architecture:



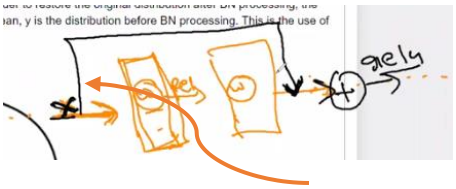
Here assume I have some dataset of x , after x I have some weight layer then relu layer and then again weight layer and relu layer. Here weight layer is nothing but the kernel.



So what if I have some negative input or negative data from the previous one

So every time instead of training the weight or it is not appropriate for this network, so in that case instead of training the particular network I will try to skip it, and then they have used + operator and then relu.

So if there is negative weight then I will try to skip those negative weights and I will not try to learn those negative weights and whatever previous layer output I will directly try to add those output as an input to the next connection.



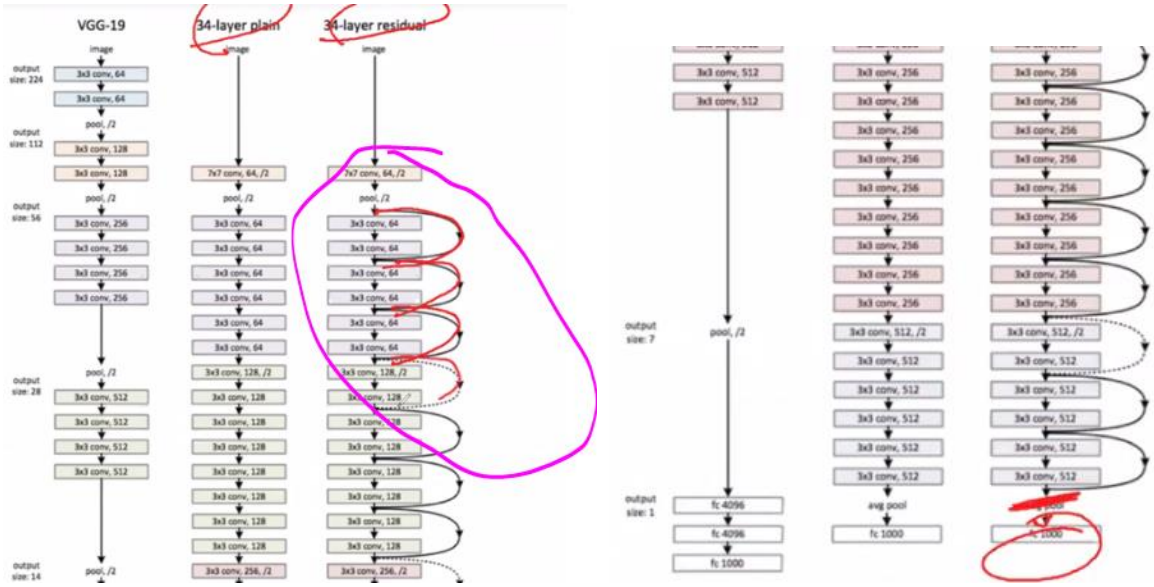
So this is called as **Skip layer or identity layer**, which means that whatever input that we are able to get same thing you are trying to consider as an output that is called as an identity layer and in between you are not trying to learn anything or change any input in between.

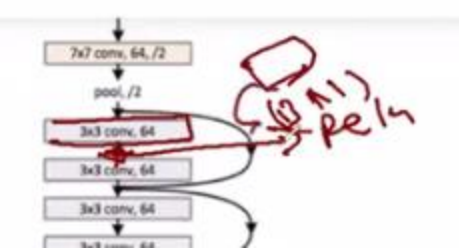
And then they we are trying to pass the Relu layer. So if some negative values in the x which means the output which I get from the previous layer that it will be remove those negative values and it will consider only the maximum of the dataset will be considered. Even in the back propagation it will skip the particular one it will not try to learn anything which is available in between.

The network layer which is not trying to learn, so that layer is called as **residual network.** That I will try to skip.

Here we take a block of two 3X3 and 3X3

But if this layer is trying to give me some kind of significant value or positive value, in that case I will try to consider the residual network and I will not skip it.

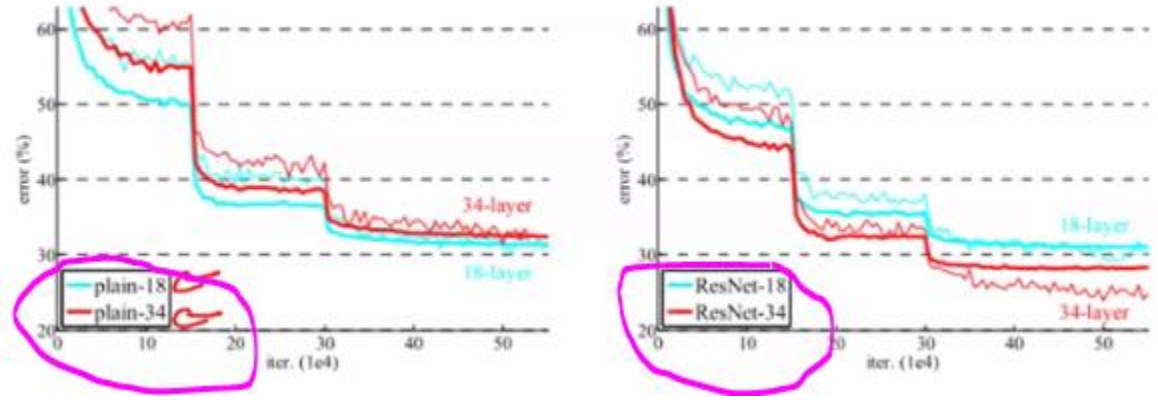




Convolution , batch normalization and relu - in between

And finally I will flatten my dataset and I send it to the feed forward network.

By this type I can reduce the number of learnable parameters, reduce the processing time, in back propagation again I will not go to each and every layer and have to trained the weight because I have skipped some of the layers.

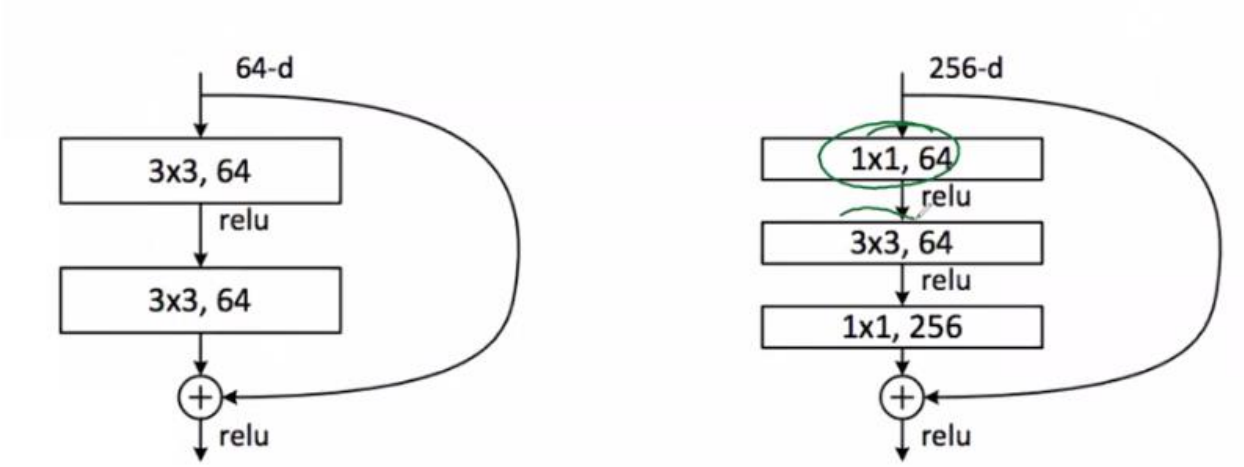


In image 1 plain 18 is performing good than 34 layer

In the second image ResNet 34 is performing much better when compare to ResNet 18.

If I have a lots and lots of features in my images so in that case I will be able to use ResNet 34 layer network.

Variants:



Diffrent Variants : -

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Error rate:

method	top-5 err. (test)
VGG [40] (ILSVRC'14)	7.32
GoogLeNet [43] (ILSVRC'14)	6.66
VGG [40] (v5)	6.8
PReLU-net [12]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

Que and clarification:

Code implementation:

Code for data augmentation:

```
import numpy as np


def get_random_eraser(p=0.5, s_l=0.02, s_h=0.4, r_1=0.3, r_2=1/0.3, v_l=0, v_h=255, pixel_level=False):
    def eraser(input_img):
        img_h, img_w, img_c = input_img.shape
        p_1 = np.random.rand()

        if p_1 > p:
            return input_img

        while True:
            s = np.random.uniform(s_l, s_h) * img_h * img_w
            r = np.random.uniform(r_1, r_2)
            w = int(np.sqrt(s / r))
            h = int(np.sqrt(s * r))
            left = np.random.randint(0, img_w)
            top = np.random.randint(0, img_h)

            if left + w <= img_w and top + h <= img_h:
                break

        if pixel_level:
            c = np.random.uniform(v_l, v_h, (h, w, img_c))
        else:
            c = np.random.uniform(v_l, v_h)

        input_img[top:top + h, left:left + w, :] = c

        return input_img

    return eraser
```

```
from tensorflow.keras.applications.resnet50 import ResNet50, preprocess_input
import tensorflow.keras as keras
from tensorflow.keras import models
from tensorflow.keras import layers
from tensorflow.keras import optimizers
import tensorflow as tf
from keras.utils import np_utils
from keras.models import load_model
from keras.datasets import cifar10
from keras.preprocessing import image
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import cv2
```

```
conv_base = ResNet50(weights='imagenet', include_top=False, input_shape=(200, 200, 3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94773248/94765736 [==============================] - 1s 0us/step
```

```
[ ] conv_base.summary()
```

conv2_block2_2_bn (BatchNormali	(None, 50, 50, 64)	256	conv2_block2_2_conv[0][0]
conv2_block2_2_relu (Activation	(None, 50, 50, 64)	0	conv2_block2_2_bn[0][0]
conv2_block2_3_conv (Conv2D)	(None, 50, 50, 256)	16640	conv2_block2_2_relu[0][0]
conv2_block2_3_bn (BatchNormali	(None, 50, 50, 256)	1024	conv2_block2_3_conv[0][0]
conv2_block2_add (Add)	(None, 50, 50, 256)	0	conv2_block1_out[0][0] conv2_block2_3_bn[0][0]
conv2_block2_out (Activation)	(None, 50, 50, 256)	0	conv2_block2_add[0][0]
conv2_block3_1_conv (Conv2D)	(None, 50, 50, 64)	16448	conv2_block2_out[0][0]
conv2_block3_1_bn (BatchNormali	(None, 50, 50, 64)	256	conv2_block3_1_conv[0][0]
conv2_block3_1_relu (Activation	(None, 50, 50, 64)	0	conv2_block3_1_bn[0][0]

```
                                                                  conv5_block2_3_bn[0][0]
conv5_block2_out (Activation)   (None, 7, 7, 2048)   0           conv5_block2_add[0][0]
conv5_block3_1_conv (Conv2D)    (None, 7, 7, 512)    1049088     conv5_block2_out[0][0]
conv5_block3_1_bn (BatchNormali (None, 7, 7, 512)    2048        conv5_block3_1_conv[0][0]
conv5_block3_1_relu (Activation (None, 7, 7, 512)    0           conv5_block3_1_bn[0][0]
conv5_block3_2_conv (Conv2D)    (None, 7, 7, 512)    2359808     conv5_block3_1_relu[0][0]
conv5_block3_2_bn (BatchNormali (None, 7, 7, 512)    2048        conv5_block3_2_conv[0][0]
conv5_block3_2_relu (Activation (None, 7, 7, 512)    0           conv5_block3_2_bn[0][0]
conv5_block3_3_conv (Conv2D)    (None, 7, 7, 2048)   1050624     conv5_block3_2_relu[0][0]
conv5_block3_3_bn (BatchNormali (None, 7, 7, 2048)   8192        conv5_block3_3_conv[0][0]
conv5_block3_add (Add)          (None, 7, 7, 2048)   0           conv5_block2_out[0][0]
                                                                 conv5_block3_3_bn[0][0]
conv5_block3_out (Activation)   (None, 7, 7, 2048)   0           conv5_block3_add[0][0]
==================================================================================================
Total params: 23,587,712
Trainable params: 23,534,592
Non-trainable params: 53,120
__________________________________________________________________________________________________
```

```
[ ] (x_train, y_train), (x_test, y_test) = cifar10.load_data()

    x_train = x_train / 255.0
    x_test = x_test / 255.0

    y_train = np_utils.to_categorical(y_train, 10)
    y_test = np_utils.to_categorical(y_test, 10)

    print(x_train.shape)
    print(x_test.shape)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 6s 0us/step
(50000, 32, 32, 3)
(10000, 32, 32, 3)
```

```
model = models.Sequential()
model.add(layers.UpSampling2D((2,2)))
model.add(layers.UpSampling2D((2,2)))
model.add(layers.UpSampling2D((2,2)))
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.BatchNormalization())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.BatchNormalization())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.BatchNormalization())
model.add(layers.Dense(10, activation='softmax'))

model.compile(optimizer=optimizers.RMSprop(lr=2e-5), loss='binary_crossentropy', metrics=['acc'])

history = model.fit(x_train, y_train, epochs=1, batch_size=20, validation_data=(x_test, y_test))
```

```
WARNING:tensorflow:Model was constructed with shape (None, 200, 200, 3) for input Tensor("input_1:0", shape=(None, 200, 200, 3), dtype=float32), but it was called on an input with
WARNING:tensorflow:Model was constructed with shape (None, 200, 200, 3) for input Tensor("input_1:0", shape=(None, 200, 200, 3), dtype=float32), but it was called on an input with
2500/2500 [==============================] - ETA: 0s - loss: 0.1975 - acc: 0.5944WARNING:tensorflow:Model was constructed with shape (None, 200, 200, 3) for input Tensor("input_1:
2500/2500 [==============================] - 715s 286ms/step - loss: 0.1975 - acc: 0.5944 - val_loss: 0.0840 - val_acc: 0.8684
```

```
[ ] #model = tf.keras.models.load_model('my_model.h5')
    model.save("my_model.h5")
```

```
[ ] model1 = tf.keras.models.load_model('my_model.h5')
```

```
#model = tf.keras.models.load_model('my_model.h5')
model.save("my_model.h5")
```

```
model1 = tf.keras.models.load_model('my_model.h5')
```

```
WARNING:tensorflow:Model was constructed with shape (None, 200, 200, 3) for input Tensor("input_1_1:0", shape=(None, 200, 200, 3), dtype=f
```

```
!pwd
```

```
/content
```

```
# example of loading an image with the Keras API
from keras.preprocessing import image
img = cv2.imread("test.jpg")
dim = (32, 32)
img = cv2.resize(img, dim, interpolation=cv2.INTER_AREA)
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)
preds = model.predict(x)
```

```
WARNING:tensorflow:Model was constructed with shape (None, 200, 200, 3) for input Tensor("input_1:0", shape=(None, 200, 200, 3), dtype=flo
```

```
preds
```

```
array([[0., 1., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)
```