1. **Convolutional layer: Write code that instantiates a 5-by-5 kernel of 32 filters for the image, and perform a 2d convolution of the image with stride 1 in each direction. Define a bias variable of shape [32] for the output of the kernel, and add the bias to the output of the 2d convolution. What will be the output tensor dimensions if we had used 64 filters of 5-by-5 and a stride of 1? What will it be if we used 32 filters of 7-by-7 and a stride of 2?**

Output tensor Dimension with 64 filters of 5-by-5 and a stride of 1:

Let's define

O = Size (width) of output image.
I = Size (width) of input image.
K = Size (width) of kernels used in the Conv Layer.
N = Number of kernels.
S = Stride of the convolution operation.
P = Padding.

The size (O) of the output tensor is given by

$$O = \frac{I - K + 2P}{S} + 1$$

The number of channels in the output image is equal to the number of kernels.

In our case,
I = 32
K = 5
N = 64
S = 1
P = 1

Therefore O=(32-5+2)/1+1=30

Number of channels=64

So the tensor matrix= 64 x [30]

What will it be if we used 32 filters of 7-by-7 and a stride of 2?

Therefore O= (32-7+2)/2+1=18.5 ~ 19 (Automatic Round off by Tensorflow)

Number of channels=32

So the tensor matrix= 32 x [19]

4. Learning: Use cross entropy as the loss, as you've done before, and train the model. Plot the training and validation loss, as well as the training and validation accuracy over 50 epochs of training. Run on 3 different hyperparameter settings (i.e. change the learning rate, weight decay coefficient, dropout) and report your results.

<u>The accuracy results for all parameters are as follows:</u>

##### MODEL PARAMETERS-5 learning_rate = 0.001 num_steps = 2000 batch_size = 32 #####Testing Accuracy: 0.15053764

##### MODEL PARAMETERS-1 learning_rate = 0.001 num_steps = 2000 batch_size = 32 #####Testing Accuracy: {'accuracy': 0.16129032, 'loss': 1.8621702, 'global_step': 0}

##### MODEL PARAMETERS-2 learning_rate = 0.0001 num_steps = 5000 batch_size = 32 #####Testing Accuracy: {'accuracy': 0.17204301, 'loss': 1.8332964, 'global_step': 0}

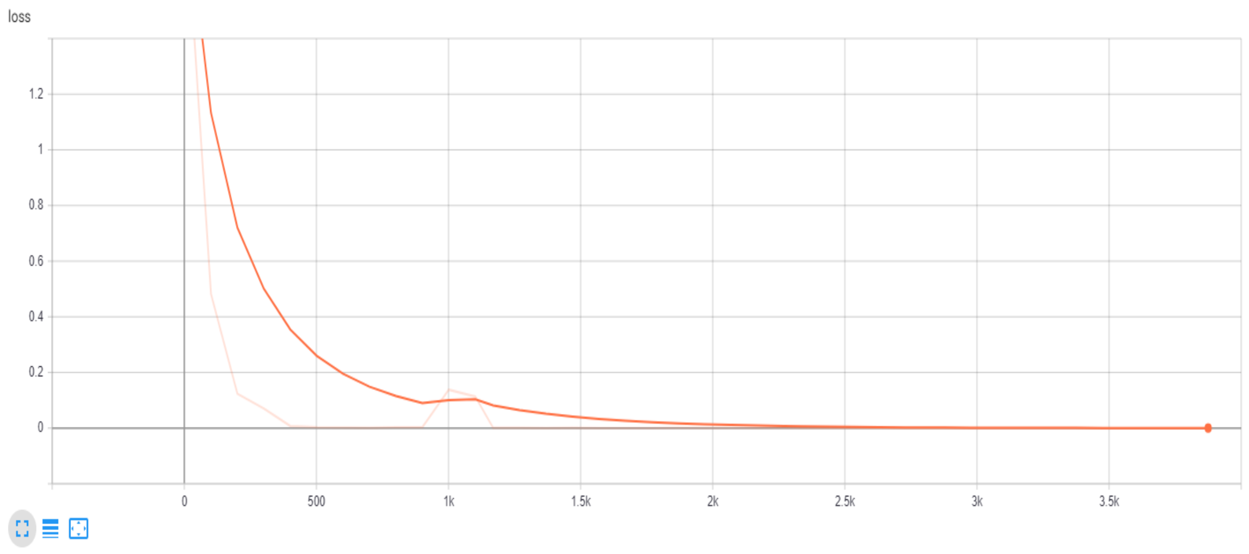
##### MODEL PARAMETERS-3 learning_rate = 0.001 num_steps = 2000 batch_size = 128 #####Testing Accuracy: {'accuracy': 0.16129032, 'loss': 1.8754494, 'global_step': 0}

##### MODEL PARAMETERS-4 learning_rate = 0.001 num_steps = 1000 batch_size = 256 #####Testing Accuracy: {'accuracy': 0.19354838, 'loss': 1.827026, 'global_step': 0}##### MODEL

PARAMETERS-5 learning_rate = 0.001 num_steps = 2000 batch_size = 32 #####Testing Accuracy: {'accuracy': 0.15053764, 'loss': 1.8067622, 'global_step': 0}

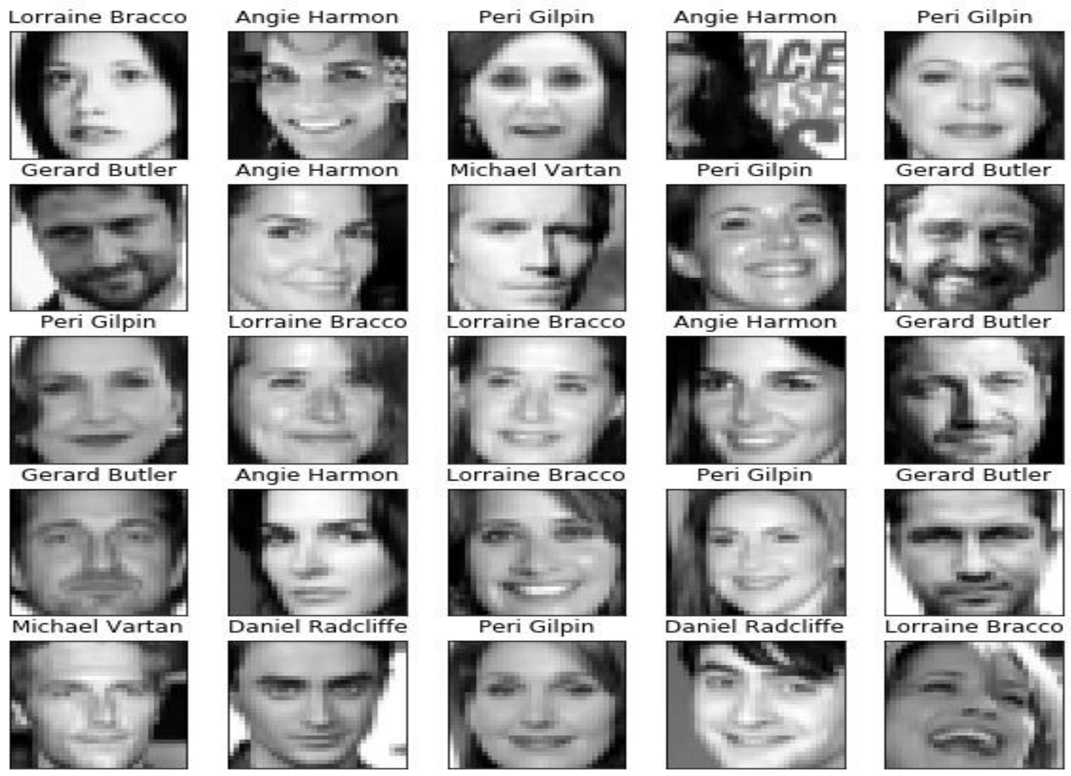
It is clear from the above results that no matter what, with only one conv layer, the test accuracy never increases beyond 20%. More conv layer is thus required. Though it is also evident that the dropout layer did its part in restricting overfitting.

<u>The average loss during training for all parameter changes is as follows:</u>

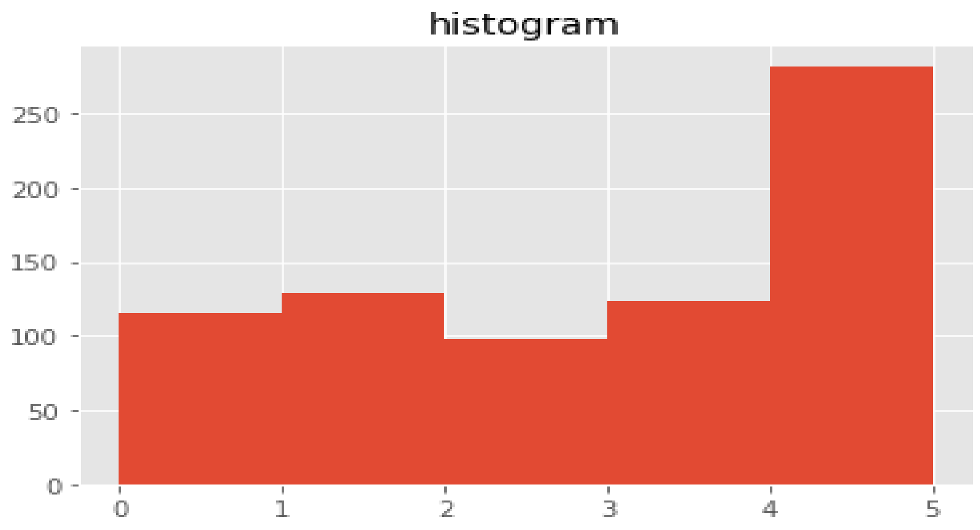


5. Visualization: To get more insight into what a convolutional neural network achieves with its architecture, you will visualize the function that the convolutional layer provides. Visu-alize 8 of the 5-by-5 kernels trained in question 1, comment on what the network is trained to recognize with these kernels and how further layers of convolution may improve the performance.

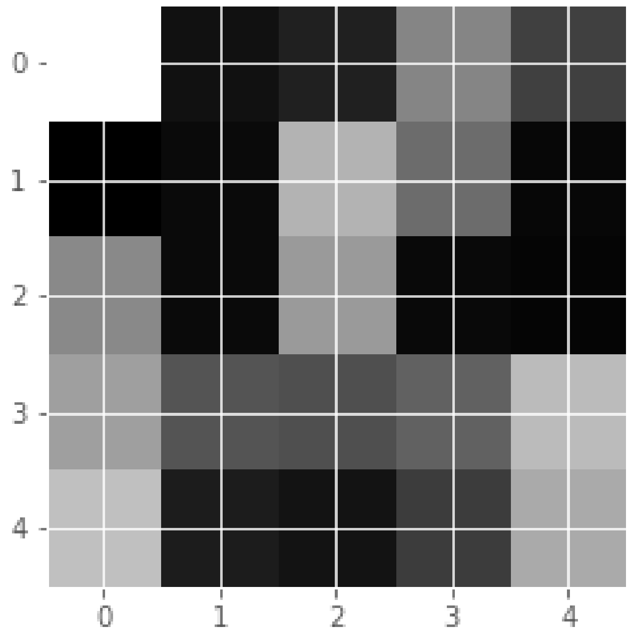
Images:



Histogram:

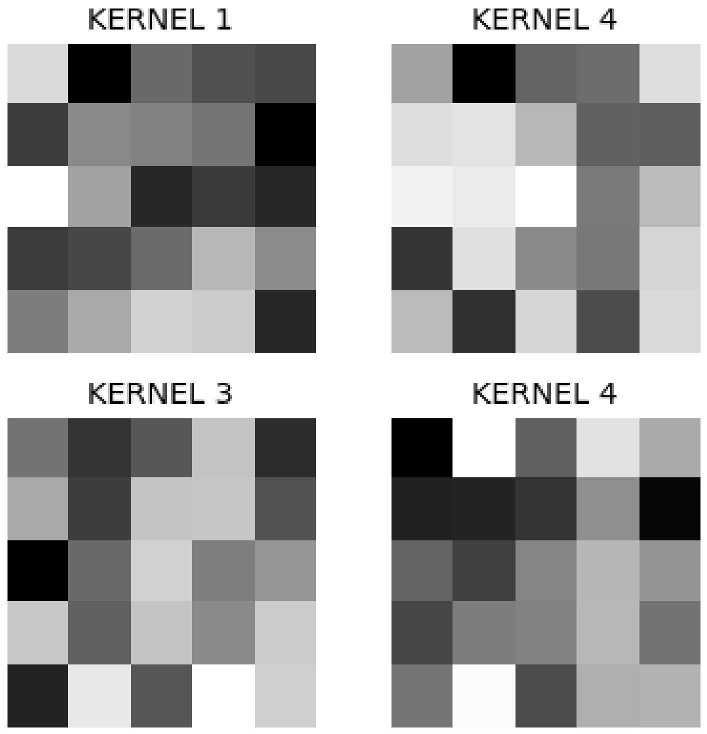


Kernels:



Kernel 1: This kernel somehow manages to create a filter which finds the outline of the face.

Let's investigate on the other kernels:



I could not any distinct features which these filters are finding. Maybe a different visualization technique would be beneficial.

Python Code for the assignment 3:

```python
# -*- coding: utf-8 -*-

import time
import math
import random

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import cv2

from sklearn.metrics import confusion_matrix
from datetime import timedelta
import os
os.environ["CUDA_VISIBLE_DEVICES"]="-1" #for training on gpu/cpu
print(tf.__version__)
data_path="data.npy"
target_path="target.npy"

print("#####################")
print("ENTER CHOICE")
print("1. Task 0----ENTER 0")
print("2. Task 1----ENTER 1")
g = input("Your Choice : ")
g=int(g)

if g==0:
    task=0
else:
    task=1

print("#####################")
print()
print("Task {0} Starts".format(task))

### LOADING OF DATA
def data_segmentation(data_path, target_path, task):
    # task = 0 >> select the name ID targets for face recognition task
    # task = 1 >> select the gender ID targets for gender recognition task
    data = np.load(data_path)/255
    data = np.reshape(data, [-1, 32*32])
    target = np.load(target_path)
    np.random.seed(45689)
    rnd_idx = np.arange(np.shape(data)[0])
    np.random.shuffle(rnd_idx)
    trBatch = int(0.8*len(rnd_idx))
    validBatch = int(0.1*len(rnd_idx))
    trainData, validData, testData = data[rnd_idx[1:trBatch],:], \
                                    data[rnd_idx[trBatch+1:trBatch + validBatch],
:],\
                                    data[rnd_idx[trBatch + validBatch+1:-1],:]
    trainTarget, validTarget, testTarget = target[rnd_idx[1:trBatch], task], \
                                            target[rnd_idx[trBatch+1:trBatch + va
lidBatch], task],\
                                            target[rnd_idx[trBatch + validBatch +
 1:-1], task]
    return trainData, validData, testData, trainTarget, validTarget, testTarget

trainData, validData, testData, trainTarget, validTarget, testTarget=data_segment
ation(data_path, target_path, task)

##  The name (ID) of the actors: 'Lorraine Bracco', 'Gerard Butler',
#'Peri Gilpin', 'Angie Harmon', 'Daniel Radcliffe', and 'Michael Vartan'
# are encoded as '0', '1', '2', '3', '4', and '5',respectively.
num_input = 1024
if task==0:
    class_names = ["Lorraine Bracco", "Gerard Butler", "Peri Gilpin", "Angie Harm
on", "Daniel Radcliffe","Michael Vartan"]
    num_classes = 6
else:
    class_names = ["Male","Female"]
    num_classes = 2

fig=plt.figure(figsize=(10, 10))
columns = 5
rows = 5
for i in range(1, columns*rows +1):
    img = trainData[i-1]*255
    ax1 = fig.add_subplot(rows, columns, i)
    ax1.set_title(class_names[trainTarget[i-1]])
    ax1.get_xaxis().set_visible(False)
    ax1.get_yaxis().set_visible(False)
    plt.imshow(img.reshape((32, 32)), cmap='gray')
    plt.grid
plt.show()
```

```
### You are provided withtwo .npy files which have 936 rows of images and labels,
 and you should divide
## the dataset into 80/10/10% for training, validation and test, respectively.
print("#####   The Data Split   ######")
print("- Training-set:\t\t{}".format(len(trainData)))
print("- Test-set:\t\t{}".format(len(testData)))
print("- Validation-set:\t{}".format(len(validData)))

##Distribution
print("##### DISTRIBUTION OF DATA #####")
a = trainTarget
plt.style.use('ggplot')
plt.hist(a, bins = [0,1,2,3,4,5])
plt.title("histogram")
plt.show()

print("##### MODEL TRAIN TASK-{} #####".format(task))
# Define the model function (following TF Estimator Template)
# Create the neural network
def conv_net(x_dict, n_classes, dropout, reuse, is_training):
    # Define a scope for reusing the variables
    with tf.variable_scope('ConvNet', reuse=reuse):

        # Data input is a 1-D vector of 1024 features (32*32 pixels)
        # Reshape to match picture format [Height x Width x Channel]
        # Tensor input become 4-D: [Batch Size, Height, Width, Channel]
        x = tf.reshape(x_dict, shape=[-1, 32, 32, 1])

        # Convolution Layer with 32 filters and a kernel size of 5
        conv1 = tf.layers.conv2d(x, 32, 5, activation=tf.nn.relu)
        # Max Pooling (down-sampling) with strides of 2 and kernel size of 3
        conv1 = tf.layers.max_pooling2d(conv1, 3, 2)


        # Flatten the data to a 1-D vector for the fully connected layer
        fc1 = tf.contrib.layers.flatten(conv1)

        # Fully connected layer (in tf contrib folder for now)
        fc1 = tf.layers.dense(fc1, 384)

        # Fully connected layer (in tf contrib folder for now)
        fc2 = tf.layers.dense(fc1, 192)



        # Apply Dropout (if is_training is False, dropout is not applied)
        fc3 = tf.layers.dropout(fc2, rate=dropout, training=is_training)

        # Output layer, class prediction
        out = tf.layers.dense(fc3, n_classes)

    return out

# Define the model function (following TF Estimator Template)
def model_fn(features, labels, mode):

    # Build the neural network
    # Because Dropout have different behavior at training and prediction time, we
    # need to create 2 distinct computation graphs that still share the same weig
hts.
    logits_train = conv_net(features, num_classes, dropout, reuse=False,
                            is_training=True)
    logits_test = conv_net(features, num_classes, dropout, reuse=True,
                           is_training=False)

    # Predictions
    pred_classes = tf.argmax(logits_test, axis=1)
    pred_probas = tf.nn.softmax(logits_test)

    # If prediction mode, early return
    if mode == tf.estimator.ModeKeys.PREDICT:
        return tf.estimator.EstimatorSpec(mode, predictions=pred_classes)

        # Define loss and optimizer
    loss_op = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(
        logits=logits_train, labels=tf.cast(labels, dtype=tf.int32)))
    optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
    train_op = optimizer.minimize(loss_op,
                                  global_step=tf.train.get_global_step())

    # Evaluate the accuracy of the model
    acc_op = tf.metrics.accuracy(labels=labels, predictions=pred_classes)

    # TF Estimators requires to return a EstimatorSpec, that specify
    # the different ops for training, evaluating, ...
    estim_specs = tf.estimator.EstimatorSpec(
        mode=mode,
        predictions=pred_classes,
        loss=loss_op,
        train_op=train_op,
        eval_metric_ops={'accuracy': acc_op})
    return estim_specs

def train():
    # Define the input function for training
    input_fn = tf.estimator.inputs.numpy_input_fn(
        x=trainData, y=trainTarget,
        batch_size=batch_size, num_epochs=50, shuffle=True)

    # Define the input function for validation
    valid_fn = tf.estimator.inputs.numpy_input_fn(
        x=validData, y=validTarget,
        batch_size=batch_size, num_epochs=50, shuffle=True)

    # Train the Model
    classifier = tf.estimator.Estimator(
        model_fn=model_fn,)

    train_spec = tf.estimator.TrainSpec(
        input_fn = input_fn,
    )

    eval_spec = tf.estimator.EvalSpec(
        input_fn = valid_fn,
        throttle_secs=120,
        start_delay_secs=120,
    )

    tf.estimator.train_and_evaluate(
        classifier,
        train_spec,
        eval_spec
    )

def evaluate():
        # Evaluate the Model
    # Define the input function for evaluating
    input_fn = tf.estimator.inputs.numpy_input_fn(
        x=testData, y=testTarget,
        batch_size=batch_size, shuffle=False)
    # Use the Estimator 'evaluate' method
    e = model.evaluate(input_fn)
    return e

acc=[]
```

```
#########   EVALUATION 1  ######
# Training Parameters
learning_rate = 0.001
num_steps = 2000
batch_size = 32
dropout = 0.5 # Dropout, probability to drop a unit
# Build the Estimator and Train
model = tf.estimator.Estimator(model_fn)
train()
e=evaluate()
print()
print("##### MODEL PARAMETERS-
1 learning_rate = 0.001 num_steps = 2000 batch_size = 32  #####")
print("Testing Accuracy:", e['accuracy'])
acc.append(e)

#########   EVALUATION 2  ######
# Training Parameters
learning_rate = 0.0001
num_steps = 5000
batch_size = 32
dropout = 0.5 # Dropout, probability to drop a unit
# Build the Estimator and Train
model = tf.estimator.Estimator(model_fn)
train()
e=evaluate()
print()
print("##### MODEL PARAMETERS-
2 learning_rate = 0.0001 num_steps = 5000 batch_size = 32  #####")
print("Testing Accuracy:", e['accuracy'])
acc.append(e)

#########   EVALUATION 3  ######
# Training Parameters
learning_rate = 0.001
num_steps = 2000
batch_size = 128
dropout = 0.5 # Dropout, probability to drop a unit
# Build the Estimator and Train
model = tf.estimator.Estimator(model_fn)
train()
e=evaluate()
print()
print("##### MODEL PARAMETERS-
3 learning_rate = 0.001 num_steps = 2000 batch_size = 128  #####")
print("Testing Accuracy:", e['accuracy'])
acc.append(e)

#########   EVALUATION 4  ######
# Training Parameters
learning_rate = 0.001
num_steps = 1000
batch_size = 256
dropout = 0.5 # Dropout, probability to drop a unit
# Build the Estimator and Train
model = tf.estimator.Estimator(model_fn)
train()
e=evaluate()
print()
print("##### MODEL PARAMETERS-
4 learning_rate = 0.001 num_steps = 1000 batch_size = 256  #####")
print("Testing Accuracy:", e['accuracy'])
acc.append(e)

#########   EVALUATION 5  ######
# Training Parameters
learning_rate = 0.001
num_steps = 2000
batch_size = 32
dropout = 0.2 # Dropout, probability to drop a unit
# Build the Estimator and Train
model = tf.estimator.Estimator(model_fn)
train()
e=evaluate()
print()
print("##### MODEL PARAMETERS-
5 learning_rate = 0.001 num_steps = 2000 batch_size = 32  #####")
print("Testing Accuracy:", e['accuracy'])
acc.append(e)

print()
print("##### MODEL PARAMETERS-
1 learning_rate = 0.001 num_steps = 2000 batch_size = 32  #####")
print("Testing Accuracy:", acc[0])
print()
print("##### MODEL PARAMETERS-
2 learning_rate = 0.0001 num_steps = 5000 batch_size = 32  #####")
print("Testing Accuracy:", acc[1])
print()
print("##### MODEL PARAMETERS-
3 learning_rate = 0.001 num_steps = 2000 batch_size = 128  #####")
print("Testing Accuracy:", acc[2])
print()
print("##### MODEL PARAMETERS-
4 learning_rate = 0.001 num_steps = 1000 batch_size = 256  #####")
print("Testing Accuracy:", acc[3])
print()
print("##### MODEL PARAMETERS-
5 learning_rate = 0.001 num_steps = 2000 batch_size = 32  #####")
print("Testing Accuracy:", acc[4])
print()

### KERNELS ###
names = classifier.get_variable_names()
print("name:", names)
#for i in names:
#    print(classifier.get_variable_value(i))

weights = classifier.get_variable_value('ConvNet/conv2d/kernel')
print(weights.shape)
a=weights[0][0][0][0:25]
b=a.reshape((5,5))
plt.imshow(b, cmap='gray')
plt.show()

names = ["KERNEL 1", "KERNEL 2", "KERNEL 3", "KERNEL 4"]

fig=plt.figure(figsize=(6, 6))
columns = 2
rows = 2
for i in range(1, columns*rows +1):
    img = trainData[i-1]*255
    ax1 = fig.add_subplot(rows, columns, i)
    ax1.set_title(names[trainTarget[i-1]])
    ax1.get_xaxis().set_visible(False)
    ax1.get_yaxis().set_visible(False)
    a=weights[i][0][0][0:25]
    b=a.reshape((5,5))
    plt.imshow(b, cmap='gray')
    plt.grid
plt.show()
```