23 - April

# A Deep Reinforcement Learning Framework For Identifying Funny Scenes In Movie

**DA 671** – Introduction to Reinforcement Learning

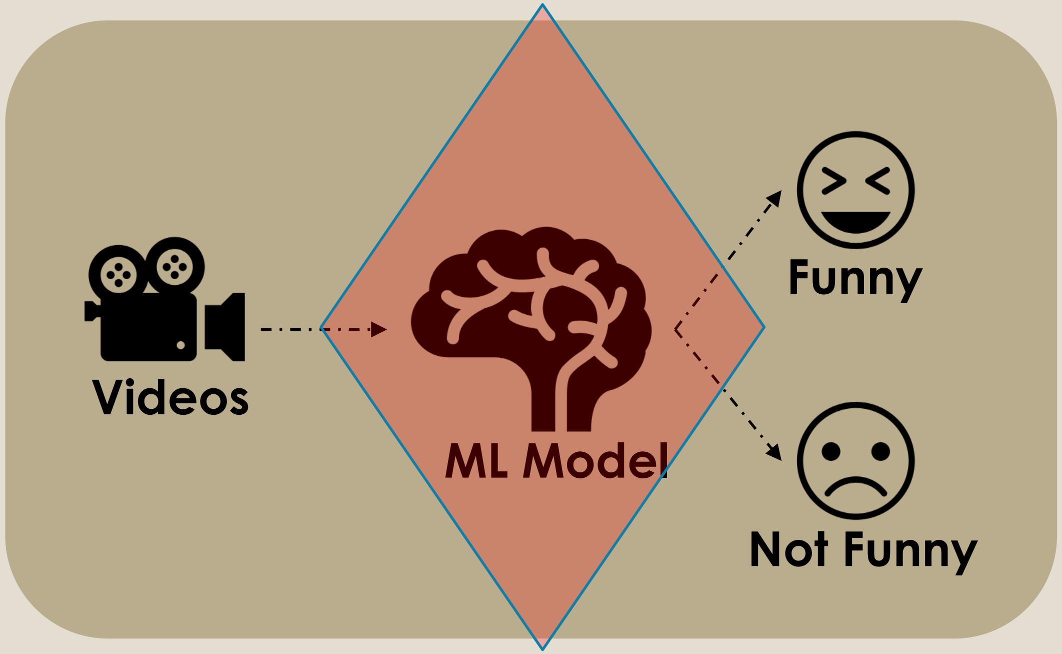
**Instructor** – Dr. Arghyadip Roy

**Team Members**

Ashish Kumar (22415600**3**)
Atul Bhagat (22415600**4**)
Ashish Goswami (2241560**15**)

# What this project is about?

- Goal is to classify movie scenes
- Extracting **affective** information from video
  - Complex Spatial and Temporal dependencies integrated with human perception and information
  - Complex non-linear Process
  - Scene level affective labels convey cumulative information from different modalities
- Efficient search and recommendations




Needs labeled data

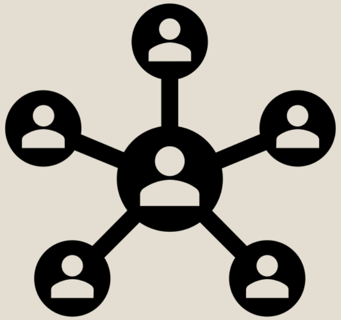
# Why use Reinforcement Learning?

- Difficult to collect a *large annotated corpus*
- Despite the large amount of available movie data, the amount of accurately labeled data is severely limited due to **copyright** and cost of annotation
- Need a learning framework based on RL that is **tolerant to label sparsity**
- Learner should easily make use of any available ground truth in an online fashion

**State Transitions and Rewards Tuple (s(t), A(t), s(t + 1), R(t)**

# Challenges with this approach?

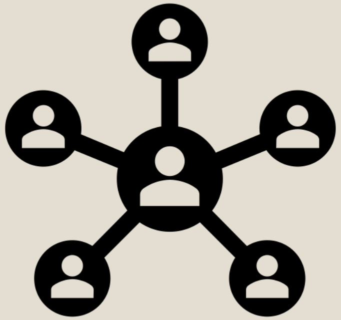
Movie Data Complexity



Agent – Environment Interaction

# Challenges with this approach?

**Movie Data Complexity**



**Agent – Environment Interaction**

**Solution**

For Simplicity, only **visual modality** is used

And, **a** parameter is used, that changes with agent's action

# Data Processing – Extracting Frames

We used **FRIENDS** TV Series to curate a data set for us

**Ffmpeg** – Unix command line tool used to extract frames

Output was rescaled and converted to grayscale

For more information https://ffmpeg.org/

```bash
#!/bin/bash

# Define the input directory and output directory
input_dir="scenes"
output_dir="frames"
wid=256
ht=-1

# Loop over each subdirectory in the input directory
for sub_dir in "$input_dir"/*/; do
  # Extract the subdirectory name (i.e., the last component of the path)
  sub_dir_name="$(basename "$sub_dir")"

  # Loop over each scene file in the subdirectory
  for scene_file in "$sub_dir"/*.mkv; do
    # Extract the basename of the scene file (without the extension)
    scene_basename="$(basename "${scene_file%.*}")"

    # Create a subdirectory in the output directory for the frames of this scene
    mkdir -p "$output_dir/$sub_dir_name/$scene_basename"

    # Use FFmpeg to extract one frame per second from the scene file
    ffmpeg -i "$scene_file" -vf "scale=$wid:$ht,format=gray,fps=20" -r 5 "$output_dir/$sub_dir_name/
$scene_basename/frame_%003d.png"
  done
done
```





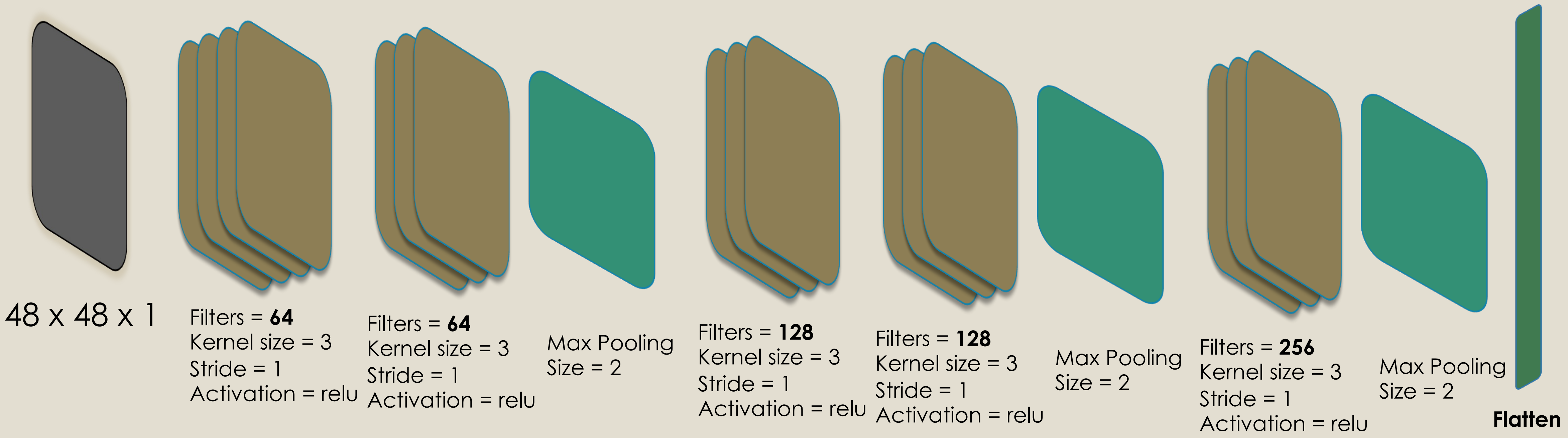

# Data Processing – Faces Extraction

On the image channel, **most affective information** is contained in the **faces** present in the frame

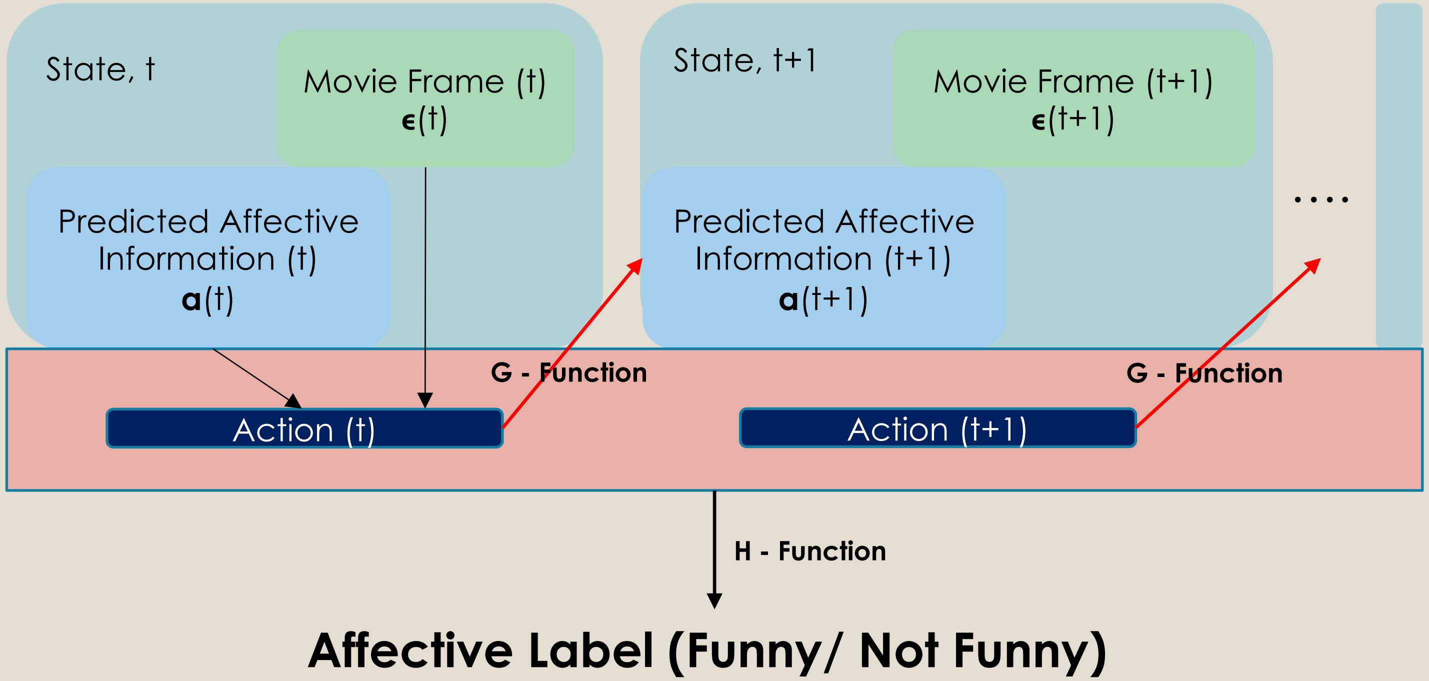
**Dlib** – Standard Face Detection Library used to extract faces

For multiple faces, one **closest to center** is only considered (assuming, main character is present in the center, thus containing *most relevant affective information*

# Facial Expressions Embeddings : CNN



# Coming to the Framework..



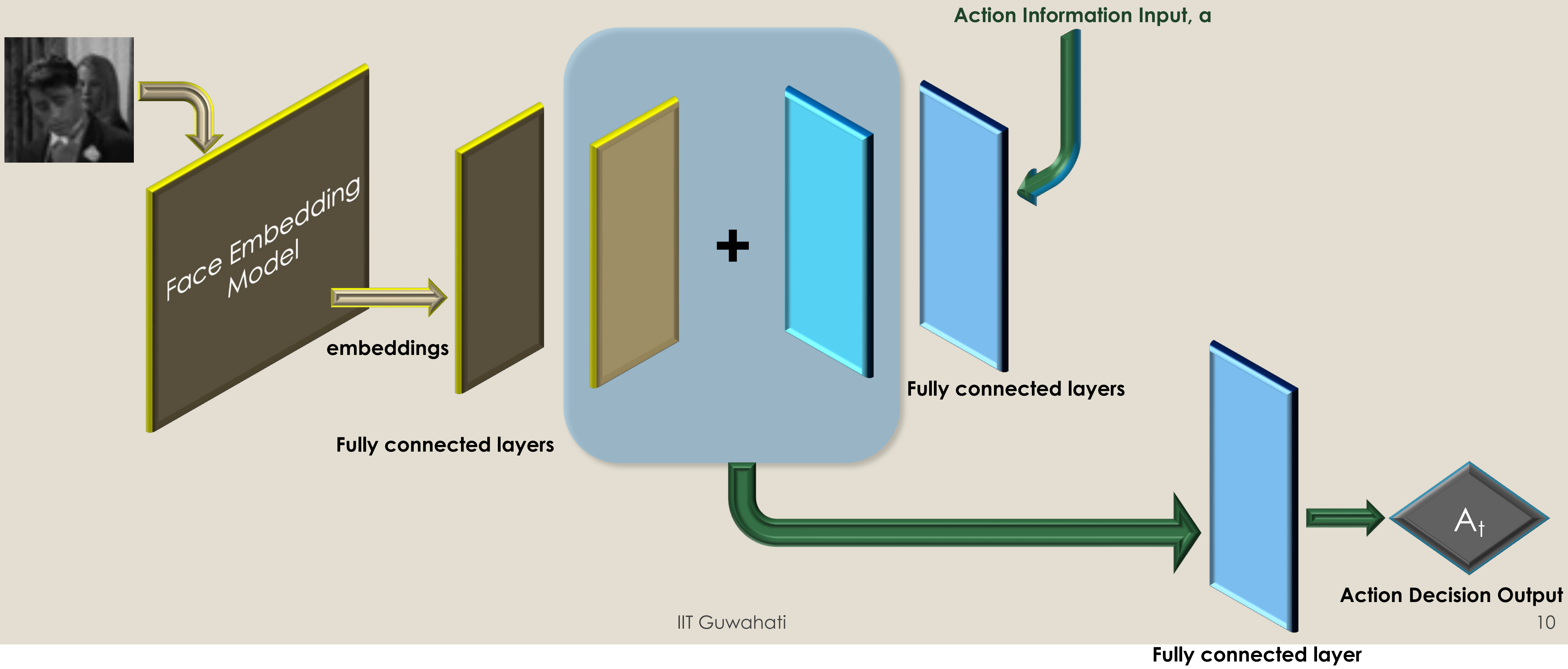

$$\mathbf{A}(t) = \mathbf{Q}(s(t)) = \mathbf{Q}(\ [\boldsymbol{\epsilon}(t), \mathbf{a}(t)]\ )$$

$$\mathbf{R}(t) = \mathbf{H}(\ A(1), \ldots, A(t), \mathbf{F}\ )$$

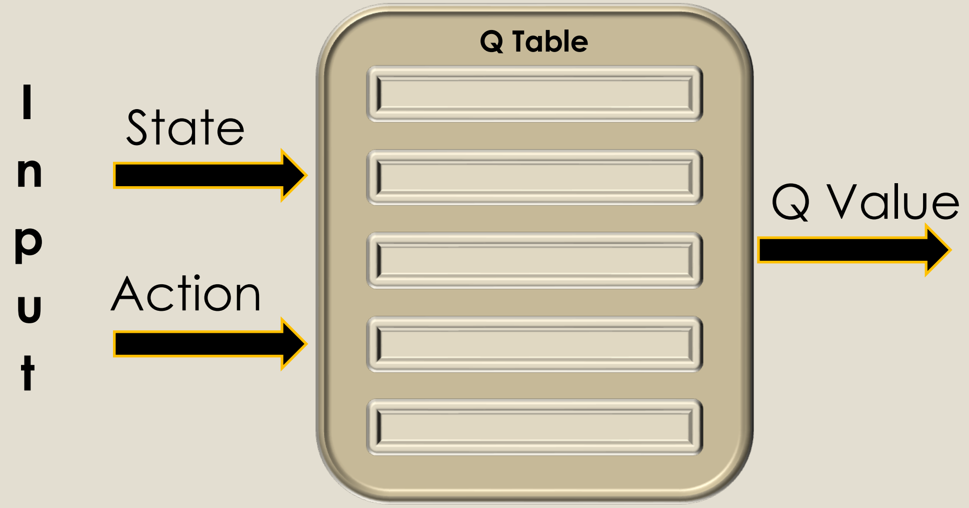
$$\mathbf{a}(t+1) = \mathbf{G}(A(1), \ldots, A(t))$$

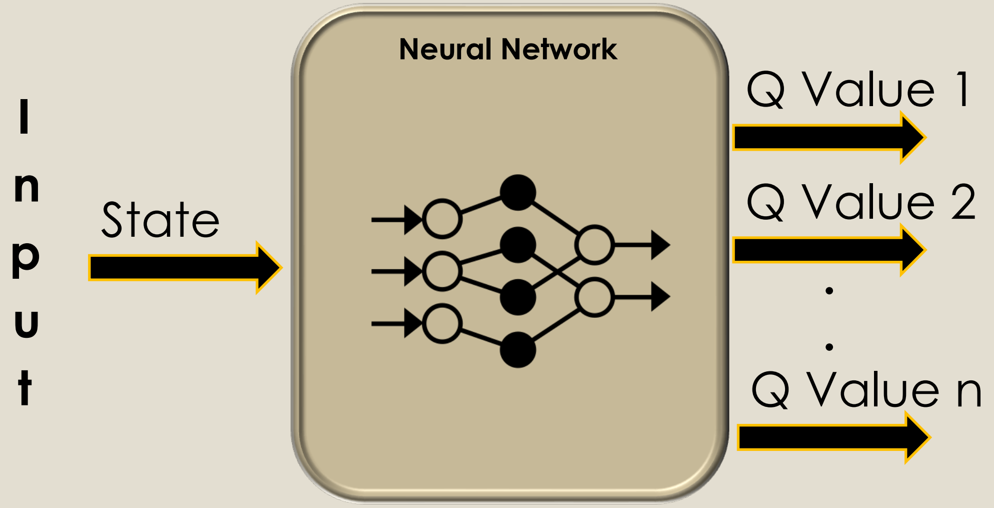
**Q** – Deep Q Network
**F** – True Affective Label
**A**(t) – Action at timestep, t
Reward Generated by **H** function

# Framework Continued…

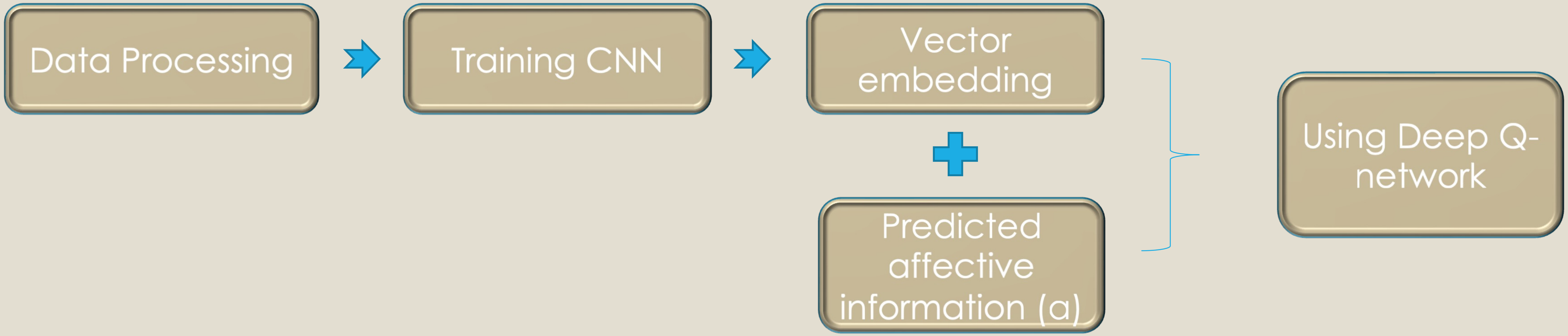


# Q Learning & DQN




**Loss Function** = [ $R_{t+1} + \gamma$ * **max** $Q(s_{t+1}, a)$ ] - **Q** $(s_t, a_t)$




- 7 classes of emotion
- Frames from video clips using dlib library

- 5 convolution layer (64,128,256)
- 20 epochs

- At each time t, vector embedding and
- Predicted affective information is used.

- Agent make decision at time t
- For input a (t+ 1) at time (t+1) either the action output at time t or human annotations at time t if available

# Let’s have a look at the code : **CNN**

```
from tensorflow.keras.layers import BatchNormalization
model=Sequential()

model.add(Conv2D(filters=64, kernel_size = (3,3), activation="relu", input_shape=(48,48,1)))
model.add(Conv2D(filters=64, kernel_size = (3,3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())

model.add(Conv2D(filters=128, kernel_size = (3,3), activation="relu"))
model.add(Conv2D(filters=128, kernel_size = (3,3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())

model.add(Conv2D(filters=256, kernel_size = (3,3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Flatten())
model.add(Dense(512,activation="relu"))

model.add(Dense(7,activation="softmax"))
```

```
model.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accuracy'])
```

```
model.summary()
```

***Note**: Only snippets are shown here*

# Let's have a look at the code : **DLib**

```
import dlib
import cv2

def cropface(frame):
    detector = dlib.get_frontal_face_detector()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = detector(gray, 1)

    if len(faces) == 0:
        return None
    elif len(faces) == 1:
        face = faces[0]
    else:
        center_x, center_y = frame.shape[1] // 2, frame.shape[0] // 2
        dist_to_center = [(f.center().x - center_x) ** 2 + (f.center().y - center_y) ** 2 for f in faces]
        face = faces[dist_to_center.index(min(dist_to_center))]

    left, top, right, bottom = face.left(), face.top(), face.right(), face.bottom()
    return frame[top:bottom, left:right]
```

# Let's have a look at the code : **DQN**

```python
import numpy as np
from keras.layers import Input, Dense, Concatenate
class DQN(tf.keras.Model):
  def __init__(self, action_size,affective_size, learning_rate, name='DQN'):
    super(DQN, self).__init__()
    tf.reset_default_graph()
    self.action_size = action_size
    self.learning_rate = learning_rate
    with tf.compat.v1.variable_scope(name):
      self.actions = tf.keras.Input(shape=(self.action_size),dtype=tf.float32, name='actions')
      self.affective = tf.keras.Input(shape=(affective_size),dtype=tf.float32, name='affactive')
      self.affective_fc1 = tf.layers.Dense(units=64, activation='relu')
      self.output1 = tf.layers.Dense(units=self.action_size, activation='linear')
      self.affective_fc2 = tf.layers.Dense( units=32, activation='relu')
      self.fused_fc1 = tf.layers.Dense(units=64,activation='relu')
      self.fused_fc2 = tf.layers.Dense(units=32,activation='relu')
  def call(self, embedding, action):
      action=np.reshape(action,(1,2))
      # print(actions)
      x=self.affective_fc1(action)
      x=self.affective_fc2(x)
      x=Concatenate(axis=-1)([x, embedding])
      x=self.fused_fc1(x)
      x=self.fused_fc2(x)
      # Q-value output layer
      x = self.output1(x)
      # Q-value for selected action
      Q = tf.reduce_sum(tf.multiply(x, action), axis=1)
      return Q

  def predict(self, state, affective):
      return self.session.run(self.output, feed_dict={self.inputs: state, self.affective: affective})
```

```python
class DQNAgent():

    def __init__(self, action_size=2,
                 discount=0.99,
                 eps_max=1.0,
                 eps_min=0.01,
                 eps_decay=0.995,
                 memory_capacity=5000,
                 lr=1e-3,
                 train_mode=True):

        # for epsilon-greedy exploration strategy
        self.epsilon = eps_max
        self.epsilon_min = eps_min
        self.epsilon_decay = eps_decay
        self.affective_size=2

        # for defining how far-sighted or myopic the agent should be
        self.discount = discount

        self.action_size = action_size

        # instances of the network for current policy and its target
        opt = tf.keras.optimizers.Adam(learning_rate=lr)
        self.policy_net= DQN( self.action_size, self.affective_size, lr)
        self.target_net = DQN(self.action_size,self.affective_size, lr)
        self.policy_net.compile(loss='mse', optimizer=opt)
        self.target_net.compile(loss='mse', optimizer=opt)
        # self.target_net.eval()
        self.target_net.set_weights(self.policy_net.get_weights()) # since no
        if not train_mode:
            self.policy_net.trainable =False
        # instance of the replay buffer
        self.memory = ReplayMemory(capacity=memory_capacity)
```

# Let’s have a look at the code : **DQN**

```python
def learn(self, batchsize,embedding,actions1,model,next_state):

    # select n samples picked uniformly at random from the experience replay memory, such that n=batchsize
    if len(self.memory) < batchsize:
        return
    # print(memory.pop())
    # state, action, rewards, next_states, dones = self.memory.sample(batchsize)
    # get q values of the actions that were taken, i.e calculate qpred;
    # actions vector has to be explicitly reshaped to nx1-vector
    state1=np.reshape(next_state,(48,48,3))
    img= cv2.cvtColor(state1,cv2.COLOR_BGR2GRAY)
    img = img.astype("float32")
    img=img/255.0
    embd=model.predict(np.asarray([img]))
    actions=np.zeros(2)
    action=self.select_action(next_state)
    actions[action]=1.0
    next_state, reward, done, info = env.step(actions)
    with tf.GradientTape() as tape:
        q_pred = self.policy_net.call(embedding,actions1)
        # calculate target q-values, such that yj = rj + q(s', a'), but if current state is a terminal state, then yj = rj
        q_target =self.target_net.call(embd,actions)
        q_target = tf.where(done, tf.zeros_like(q_target), q_target)
        y_j = reward + (self.discount * q_target)
        y_j = tf.reshape(y_j, (-1, 1))
        # calculate the loss as the mean-squared error of yj and qpred

        loss = tf.keras.losses.MSE(y_j, q_pred)
        print("loss=" ,loss)
    gradients = tape.gradient(loss, self.policy_net.trainable_variables)
    self.policy_net.optimizer.apply_gradients(zip(gradients, self.policy_net.trainable_variables))
```

# Let's have a look at the code : **DQN**

```python
def train(env,dqn_agent,dataset,actions,num_train_eps,update_frequency,batchsize,results_basepath):
  # print(actions,dataset[0].shape)
  fill_memory(env, dqn_agent, dataset,actions)
  print('Memory filled. Current capacity: ', len(dqn_agent.memory))

  reward_history = []
  epsilon_history = []
  step_cnt = 0
  best_score = -np.inf

  for ep_cnt in range(num_train_eps):
      epsilon_history.append(dqn_agent.epsilon)
      # print(dqn_agent.memory.sample(1)[3])
      for i in range(len(dataset)):
          done = False
          state =  env.reset(dataset[i])
          state1=np.reshape(state,(48,48,3))
          img= cv2.cvtColor(state1,cv2.COLOR_BGR2GRAY)
          img = img.astype("float32")
          img=img/255.0
          embd=embedding_model.predict(np.asarray([img]))
          ep_score = 0

          while not done:
              print(ep_cnt,i)
              action = dqn_agent.select_action(img)
              actions=np.zeros(2)
              actions[action]=1.0
              actions=actions.astype("float32")
              next_state, reward, done, info = env.step(actions)
              dqn_agent.memory.push(state=state, action=actions, next_state=next_state, reward=reward, done=done)
              dqn_agent.learn(batchsize=batchsize,embedding=embd,actions1=actions,model=embedding_model,next_state=next_state)
```

# Let's have a look at the code : **DQN**

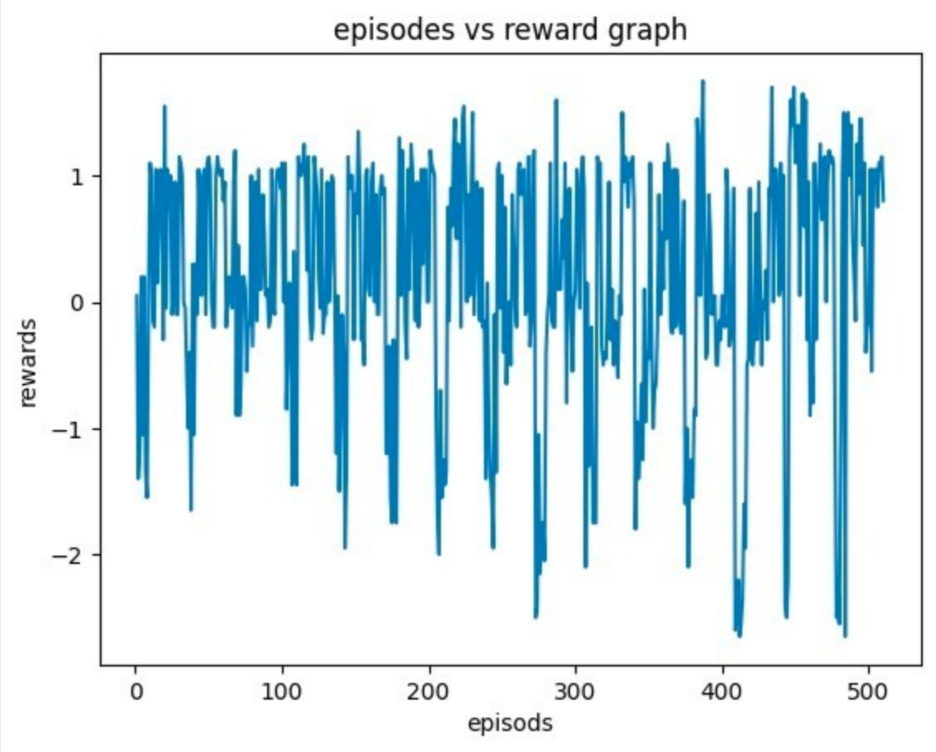
```
            if step_cnt % update_frequency == 0:
                dqn_agent.update_target_net()

            state = next_state
            ep_score += reward
            step_cnt += 1

        dqn_agent.update_epsilon()

        reward_history.append(ep_score)
        current_avg_score = np.mean(reward_history[-100:]) # moving average of last 100 episodes

        print('Ep: {}, Total Steps: {}, Ep: Score: {}, Avg score: {}; Epsilon: {}'.format(ep_cnt,step_cnt, ep_score, current_avg_score, epsilon_history[-1]))

        if current_avg_score >= best_score:
            dqn_agent.save_model('{}/dqn_model'.format(results_basepath))
            best_score = current_avg_score

with open('{}/train_reward_history.pkl'.format(results_basepath), 'wb') as f:
    pickle.dump(reward_history, f)

with open('{}/train_epsilon_history.pkl'.format(results_basepath), 'wb') as f:
    pickle.dump(epsilon_history, f)
```

# Let's have a look at the code : **DQN**

```
def train(env,dqn_agent,dataset,actions,num_train_eps,update_frequency,batchsize,results_basepath):
  # print(actions,dataset[0].shape)
  fill_memory(env, dqn_agent, dataset,actions)
  print('Memory filled. Current capacity: ', len(dqn_agent.memory))

  reward_history = []
  epsilon_history = []
  step_cnt = 0
  best_score = -np.inf

  for ep_cnt in range(num_train_eps):
      epsilon_history.append(dqn_agent.epsilon)
      # print(dqn_agent.memory.sample(1)[3])
      for i in range(len(dataset)):
          done = False
          state =  env.reset(dataset[i])
          state1=np.reshape(state,(48,48,3))
          img= cv2.cvtColor(state1,cv2.COLOR_BGR2GRAY)
          img = img.astype("float32")
          img=img/255.0
          embd=embedding_model.predict(np.asarray([img]))
          ep_score = 0

          while not done:
              print(ep_cnt,i)
              action = dqn_agent.select_action(img)
              actions=np.zeros(2)
              actions[action]=1.0
              actions=actions.astype("float32")
              next_state, reward, done, info = env.step(actions)
              dqn_agent.memory.push(state=state, action=actions, next_state=next_state, reward=reward, done=done)
              dqn_agent.learn(batchsize=batchsize,embedding=embd,actions1=actions,model=embedding_model,next_state=next_state)
```

# Model Training and Results…

```
1/1 [==============================] - 0s 31ms/step
actions= tf.Tensor([1.0866777], shape=(1,), dtype=float32) tf.Tensor([[1.0866777  0.94620126]], shape=(1, 2), dtype=float32)
1/1 [==============================] - 0s 33ms/step
actions= tf.Tensor([0.8867666], shape=(1,), dtype=float32) tf.Tensor([[0.8867666 0.7592584]], shape=(1, 2), dtype=float32)
loss= tf.Tensor([0.00751302], shape=(1,), dtype=float32)
Ep: 14, Total Steps: 7956, Ep: Score: 1.1, Avg score: 0.16949999999999996; Epsilon: 0.09199970504166631
14 32
1/1 [==============================] - 0s 27ms/step
actions= tf.Tensor([0.9435819], shape=(1,), dtype=float32) tf.Tensor([[0.9435819 0.8060961]], shape=(1, 2), dtype=float32)
1/1 [==============================] - 0s 28ms/step
actions= tf.Tensor([0.9435819], shape=(1,), dtype=float32) tf.Tensor([[0.9435819 0.8060961]], shape=(1, 2), dtype=float32)
loss= tf.Tensor([0.00115699], shape=(1,), dtype=float32)
14 32
1/1 [==============================] - 0s 29ms/step
actions= tf.Tensor([0.94522446], shape=(1,), dtype=float32) tf.Tensor([[0.94522446 0.80432075]], shape=(1, 2), dtype=float32)
1/1 [==============================] - 0s 27ms/step
actions= tf.Tensor([0.93341285], shape=(1,), dtype=float32) tf.Tensor([[0.93341285 0.7978898 ]], shape=(1, 2), dtype=float32)
loss= tf.Tensor([0.0030528], shape=(1,), dtype=float32)
14 32
1/1 [==============================] - 0s 29ms/step
actions= tf.Tensor([0.93278104], shape=(1,), dtype=float32) tf.Tensor([[0.93278104 0.7949435 ]], shape=(1, 2), dtype=float32)
1/1 [==============================] - 0s 30ms/step
actions= tf.Tensor([0.92538965], shape=(1,), dtype=float32) tf.Tensor([[0.92538965 0.7878702 ]], shape=(1, 2), dtype=float32)
loss= tf.Tensor([0.00451839], shape=(1,), dtype=float32)
14 32
1/1 [==============================] - 0s 30ms/step
actions= tf.Tensor([0.92754316], shape=(1,), dtype=float32) tf.Tensor([[0.92754316 0.7867954 ]], shape=(1, 2), dtype=float32)
1/1 [==============================] - 0s 27ms/step
actions= tf.Tensor([0.92754316], shape=(1,), dtype=float32) tf.Tensor([[0.92754316 0.7867954 ]], shape=(1, 2), dtype=float32)
loss= tf.Tensor([0.00524999], shape=(1,), dtype=float32)
Ep: 14, Total Steps: 7960, Ep: Score: 1.15, Avg score: 0.20699999999999996; Epsilon: 0.09199970504166631
14 33
```



# How to improve on this?

- Use **3D CNN**, that could capture the temporal dependency in a better fashion
- **Create a DataSet** from old movies/ TV Shows, out of the copyright
- Create a **online task** for training, like training on YouTube shorts, Instagram reels and extracting labels from the comments
- Include **audio modality** in the model
- Use a **knowledge base** to understand sarcasm/ puns etc. from subtitles, as well as audio

# THANK YOU

***Torture the data, and it will confess to anything*** ~ Ronald Coase