2025.01.15.목

생성일	@2026년 1월 15일 오전 9:12
태그	

목차

1. 티처블머신에 모델 띄워보기
2. 리액트 훅 실습

▼ 1. 티처블머신 코드

https://github.com/sanggyoon/202601-KakaoCloudAIaaS-TIL/tree/main/TestCode/20260115/react-dog-cat-model

```
import { useState, useRef, useEffect } from 'react';
// import * as tmImage from '@teachablemachine/image'; → 오랜된 라이브러리 제거
import './App.css';

const MODEL_URL = '/dogcat-model/';

function App() {
  const [isStarted, setIsStarted] = useState(false);
  const [predictions, setPredictions] = useState([]);
  const [isLoading, setIsLoading] = useState(false);

  const webcamContainerRef = useRef(null);
  const modelRef = useRef(null);
  const webcamRef = useRef(null);
  const animationRef = useRef(null);

  const init = async () => {
    if (!window.tmImage || !window.tf) {
      console.error('TM 또는 TF가 아직 로드되지 않았습니다');
      return;
    }

    setIsLoading(true);

    try {
      const tmImage = window.tmImage;

      const modelURL = MODEL_URL + 'model.json';
      const metadataURL = MODEL_URL + 'metadata.json';

      // 모델 로드
      modelRef.current = await tmImage.load(modelURL, metadataURL);
      // todo: 모델 로딩 마무리
      setIsStarted(true);
      setIsLoading(false);

      // todo: 웹캠 설정
      const flip = true;
      webcamRef.current = new tmImage.Webcam(200, 200, flip);
      await webcamRef.current.setup();
      await webcamRef.current.play();

      // todo: DOM에 웹캠 캔버스 추가
      if (webcamContainerRef.current) {
```

```
      webcamContainerRef.current.appendChild(webcamRef.current.canvas);
    }

    // todo: 초기 상태 셋업
    setPredictions([]);

    // todo: 루프 시작
    loop();
  } catch (error) {
    console.error('초기화 오류:', error);
    setIsLoading(false);
  }
};

const loop = () => {
  if (webcamRef.current) {
    webcamRef.current.update();
    predict();
  }
  animationRef.current = window.requestAnimationFrame(loop);
};

const predict = async () => {
  if (modelRef.current && webcamRef.current) {
    const prediction = await modelRef.current.predict(
      webcamRef.current.canvas
    );
    setPredictions(prediction);
  }
};

// 컴포넌트 언마운트 시 정리
useEffect(() => {
  return () => {
    if (animationRef.current) {
      window.cancelAnimationFrame(animationRef.current);
    }
    if (webcamRef.current) {
      webcamRef.current.stop();
    }
  };
}, []);

return (
  <div className="app">
    <h1>Teachable Machine Image Model</h1>

    {!isStarted && (
      <button onClick={init} disabled={isLoading}>
        {isLoading ? '로딩 중...' : 'Start'}
      </button>
    )}

    <div ref={webcamContainerRef} className="webcam-container" />

    <div className="label-container">
      {predictions.map((pred, index) => (
        <div key={index} className="prediction">
          {pred.className}: {(pred.probability * 100).toFixed(1)}%
        </div>
      ))}
    </div>
  </div>
 );
}

export default App;
```

▼ 2) React Hook

https://github.com/sanggyoon/202601-KakaoCloudAIaaS-TIL/tree/main/TestCode/20260115/react-hook

```
import { useState } from 'react';
import { useRef } from 'react';
import { useEffect } from 'react';
import './App.css';

function App() {
  // 폼 데이터 관리 ====================================================
  const [formdata, setformdata] = useState({
    username: '',
    email: '',
    password: '',
    confirmPassword: '',
  });

  const handleChange = (idname) => (event) => {
    const value = event.target.value;

    setformdata((prevState) => ({
      ...prevState,
      [idname]: value,
    }));
  };

  // Undo, Redo, History가 있는 counter ====================================================
  function useHistoryState(initialValue) {
    const pastRef = useRef([]);
    const futureRef = useRef([]);

    const [state, setState] = useState(initialValue);
    const [canUndo, setCanUndo] = useState(false);
    const [canRedo, setCanRedo] = useState(false);

    const setUpdateWithHistory = (val) => {
      pastRef.current.push(state);
      futureRef.current = [];
      setState(val);
      setCanRedo(false);
      setCanUndo(true);
    };

    const undo = () => {
      if (pastRef.current.length === 0) return;
      const previousValue = pastRef.current.pop();
      futureRef.current.push(state);
      setState(previousValue);
      setCanRedo(true);
      setCanUndo(pastRef.current.length > 0);
    };

    const redo = () ⇒ {
      if (futureRef.current.length === 0) return;
      const nextValue = futureRef.current.pop();
      pastRef.current.push(state);
      setState(nextValue);
      setCanUndo(true);
      setCanRedo(futureRef.current.length > 0);
    };

    return {
      value: state,
      setValue: setUpdateWithHistory,
      undo,
      redo,
      canUndo,
      canRedo,
    };
  }

  const { value, setValue, undo, redo, canUndo, canRedo } = useHistoryState(0);

  // 렌더링 횟수 카운터 =====================================================
  function useRenderCount() {
    // TODO: 구현하세요
    const rendering = useRef(0);
    rendering.current += 1;
    return rendering.current;
  }

  const [rand, setRand] = useState(0);

  // 사용
  // function MyComponent() {
  //   const renderCount = useRenderCount();
  //   console.log(`렌더링 횟수: ${renderCount}`);
  // }

  const renderCountNum = useRenderCount();

  // 이전 props와 변경된 props 비교 =====================================================
  function useWhyDidYouUpdate(props) {
    const previousProps = useRef(null);

    useEffect(() ⇒ {
      if (previousProps.current) {
        const changes = {};

        for (const key in props) {
          if (previousProps.current[key] !== props[key]) {
            changes[key] = {
              from: previousProps.current[key],
              to: props[key],
            };
          }
        }

        console.log('changedProps', changes);
      }
      previousProps.current = props;
    });
  }

  useWhyDidYouUpdate({ rand });

  // 사용
  // function MyComponent(props) {
  //   useWhyDidYouUpdate('MyComponent', props);
  //   // 콘솔: [MyComponent] 변경된 props: { count: { from: 1, to: 2 } }
  // }

  // useInterval 구현 ======================================================
  function useInterval(callback, delay) {
    const savedCallback = useRef();

    // 최신 콜백을 저장
    useEffect(() ⇒ {
      savedCallback.current = callback;
    }, [callback]);

    // 설정된 간격마다 콜백 실행
    useEffect(() ⇒ {
      if (delay === null) return;

      function tick() {
        savedCallback.current();
      }

      const id = setInterval(tick, delay);
      return () ⇒ clearInterval(id);
    }, [delay]);
  }

  const [count, setCount] = useState(0);
  const [delay, setDelay] = useState(1000);

  useInterval(() ⇒ {
    setCount((c) ⇒ c + 1);
  }, delay);

  // 사용 예시
  // function Timer() {
  //   const [count, setCount] = useState(0);
  //   const [delay, setDelay] = (useState < number) | (null > 1000);

  //   useInterval(() ⇒ {
  //     setCount((c) ⇒ c + 1);
  //   }, delay);
  // }

  return (
    <>
      <form
        style={{ border: '1px solid black', padding: '25px', margin: '25px' }}
      >
        <div>이름</div>
        <input
          value={formdata.username}
          onChange={handleChange('username')}
          id="username"
          type="text"
        />
        <br />
        <span>{formdata.username}</span>

        <div>이메일</div>
        <input
          value={formdata.email}
          onChange={handleChange('email')}
          id="email"
          type="email"
        />
        <br />
        <span>{formdata.email}</span>

        <div>비밀번호</div>
        <input
          value={formdata.password}
          onChange={handleChange('password')}
          id="password"
          type="password"
        />
        <br />
        <span>{formdata.password}</span>

        <div>비밀번호 확인</div>
        <input
          value={formdata.confirmPassword}
          onChange={handleChange('confirmPassword')}
          id="confirmPassword"
          type="text"
        />
        <br />
        <span>{formdata.confirmPassword}</span>
      </form>

      {/* ====================================================== */}

      <div
        style={{ border: '1px solid black', padding: '25px', margin: '25px' }}
      >
        <p>값: {value}</p>
        <button onClick={() ⇒ setValue(value + 1)}>+1</button>
        <button onClick={undo} disabled={!canUndo}>
          Undo
        </button>
        <button onClick={redo} disabled={!canRedo}>
          Redo
        </button>
      </div>

      {/* ====================================================== */}
      <div
        style={{ border: '1px solid black', padding: '25px', margin: '25px' }}
      >
        <span>렌더링 횟수: {renderCountNum / 2}</span>
        <br />
        <span>랜덤: {rand}</span>
```

```
        <br />
        <button onClick={() ⇒ setRand(Math.random())}>
          Random Number for Rerendering
        </button>
      </div>

      {/* ===================================================== */}
      <div
        style={{ border: '1px solid black', padding: '25px', margin: '25px' }}
      >
        <span>프롭스 변경은 콘솔 확인</span>
      </div>

      {/* ===================================================== */}
      <div
        style={{ border: '1px solid black', padding: '25px', margin: '25px' }}
      >
        <p>Count: {count}</p>
        <button onClick={() ⇒ setDelay(delay ? null : 1000)}>
          {delay ? '정지' : '시작'}
        </button>
        <button onClick={() ⇒ setDelay((d) ⇒ (d ? d / 2 : 1000))}>
          속도 2배
        </button>
      </div>
    </>
  );
}

export default App;
```