01-cifar10-ann

September 6, 2024

[1]:
```
import numpy as np
from keras.models import Sequential
import matplotlib.pyplot as plt

from tensorflow import keras
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
from keras.utils import to_categorical
```

1 Loading CIFAR10 Dataset for ANN Study

[2]:
```
cifar=keras.datasets.cifar10
(X_train,y_train),(X_test,y_test)=cifar.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071             4s
0us/step
```

[3]:
```
X_train=X_train/255.0
X_test=X_test/255.0
n=len(np.unique(y_train))
```

2 2 Hidden Layer Study in ANN

[4]:
```
y_train=to_categorical(y_train,n)
y_test=to_categorical(y_test,n)
model = Sequential()
model.add(Flatten(input_shape=(32, 32, 3)))
model.add(Dense(units=32,activation='relu'))
model.add(Dense(units=64,activation='relu'))
model.add(Dense(units=10,activation='softmax'))

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

```
answer=model.fit(X_train,y_train,epochs=10,verbose=2,
                 validation_split=0.2, batch_size=64)
model.evaluate(X_test,y_test)
```

```
/usr/local/lib/python3.10/dist-
packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
Epoch 1/10
625/625 - 4s - 6ms/step - accuracy: 0.1650 - loss: 2.1417 - val_accuracy: 0.1760
- val_loss: 2.0846
Epoch 2/10
625/625 - 2s - 3ms/step - accuracy: 0.1805 - loss: 2.0732 - val_accuracy: 0.1863
- val_loss: 2.0691
Epoch 3/10
625/625 - 3s - 5ms/step - accuracy: 0.1875 - loss: 2.0638 - val_accuracy: 0.1903
- val_loss: 2.0646
Epoch 4/10
625/625 - 5s - 7ms/step - accuracy: 0.1873 - loss: 2.0599 - val_accuracy: 0.1821
- val_loss: 2.0782
Epoch 5/10
625/625 - 3s - 5ms/step - accuracy: 0.1891 - loss: 2.0557 - val_accuracy: 0.1913
- val_loss: 2.0623
Epoch 6/10
625/625 - 2s - 3ms/step - accuracy: 0.1915 - loss: 2.0561 - val_accuracy: 0.1897
- val_loss: 2.0597
Epoch 7/10
625/625 - 4s - 6ms/step - accuracy: 0.1884 - loss: 2.0531 - val_accuracy: 0.1874
- val_loss: 2.0658
Epoch 8/10
625/625 - 4s - 6ms/step - accuracy: 0.1902 - loss: 2.0479 - val_accuracy: 0.1910
- val_loss: 2.0540
Epoch 9/10
625/625 - 3s - 4ms/step - accuracy: 0.1919 - loss: 2.0499 - val_accuracy: 0.1925
- val_loss: 2.0609
Epoch 10/10
625/625 - 2s - 3ms/step - accuracy: 0.1926 - loss: 2.0505 - val_accuracy: 0.1895
- val_loss: 2.0688
313/313             0s 1ms/step -
accuracy: 0.2020 - loss: 2.0574
```

[4]:
```
[2.0551371574401855, 0.19329999387264252]
```

[5]:
```
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
print(f'\nTest accuracy for 2 Hidden Layer: {test_acc:.3f}')
```

```
313/313 - 1s - 2ms/step - accuracy: 0.1933 - loss: 2.0551

Test accuracy for 2 Hidden Layer: 0.193
```

3 3 Hidden Layer Study in ANN

[6]:
```
model1 = Sequential()
model1.add(Flatten(input_shape=(32, 32, 3)))
model1.add(Dense(units=32,activation='relu'))
model1.add(Dense(units=64,activation='relu'))
model1.add(Dense(units=64,activation='relu'))
model1.add(Dense(units=10,activation='softmax'))

model1.compile(optimizer='adam',loss='categorical_crossentropy',
               metrics=['accuracy'])
answer=model1.fit(X_train,y_train,epochs=10,verbose=2,
                  validation_split=0.2,batch_size=64)
```

```
Epoch 1/10
625/625 - 4s - 6ms/step - accuracy: 0.2867 - loss: 1.9323 - val_accuracy: 0.3414
- val_loss: 1.8155
Epoch 2/10
625/625 - 4s - 7ms/step - accuracy: 0.3614 - loss: 1.7710 - val_accuracy: 0.3784
- val_loss: 1.7356
Epoch 3/10
625/625 - 4s - 6ms/step - accuracy: 0.3812 - loss: 1.7156 - val_accuracy: 0.3831
- val_loss: 1.7230
Epoch 4/10
625/625 - 2s - 4ms/step - accuracy: 0.3987 - loss: 1.6736 - val_accuracy: 0.3878
- val_loss: 1.7219
Epoch 5/10
625/625 - 3s - 4ms/step - accuracy: 0.4070 - loss: 1.6496 - val_accuracy: 0.3884
- val_loss: 1.7010
Epoch 6/10
625/625 - 2s - 3ms/step - accuracy: 0.4140 - loss: 1.6324 - val_accuracy: 0.4141
- val_loss: 1.6514
Epoch 7/10
625/625 - 2s - 3ms/step - accuracy: 0.4199 - loss: 1.6132 - val_accuracy: 0.3879
- val_loss: 1.6883
Epoch 8/10
625/625 - 4s - 6ms/step - accuracy: 0.4206 - loss: 1.6043 - val_accuracy: 0.3906
- val_loss: 1.6876
Epoch 9/10
625/625 - 2s - 3ms/step - accuracy: 0.4261 - loss: 1.5971 - val_accuracy: 0.4138
- val_loss: 1.6224
Epoch 10/10
625/625 - 3s - 4ms/step - accuracy: 0.4300 - loss: 1.5798 - val_accuracy: 0.4070
```

```
- val_loss: 1.6751
```

[7]:
```
test_loss, test_acc = model1.evaluate(X_test, y_test, verbose=2)
print(f'\nTest accuracy for 3 hidden layer: {test_acc:.3f}')
```

```
313/313 - 0s - 1ms/step - accuracy: 0.4106 - loss: 1.6494

Test accuracy for 3 hidden layer: 0.411
```

4 1 Hidden Layer Study in ANN

[8]:
```
model2 = Sequential()
model2.add(Flatten(input_shape=(32, 32, 3)))
model2.add(Dense(units=32,activation='relu'))
model2.add(Dense(units=10,activation='softmax'))
model2.compile(optimizer='adam',loss='categorical_crossentropy',
               metrics=['accuracy'])
answer=model2.fit(X_train,y_train,epochs=10,verbose=2,
                  validation_split=0.2,batch_size=64)
```

```
Epoch 1/10
625/625 - 3s - 5ms/step - accuracy: 0.2573 - loss: 2.0219 - val_accuracy: 0.2939
- val_loss: 1.9700
Epoch 2/10
625/625 - 4s - 6ms/step - accuracy: 0.3056 - loss: 1.9076 - val_accuracy: 0.3091
- val_loss: 1.9057
Epoch 3/10
625/625 - 4s - 6ms/step - accuracy: 0.3198 - loss: 1.8722 - val_accuracy: 0.3197
- val_loss: 1.8838
Epoch 4/10
625/625 - 3s - 4ms/step - accuracy: 0.3205 - loss: 1.8584 - val_accuracy: 0.3151
- val_loss: 1.8869
Epoch 5/10
625/625 - 2s - 3ms/step - accuracy: 0.3301 - loss: 1.8443 - val_accuracy: 0.3292
- val_loss: 1.8511
Epoch 6/10
625/625 - 4s - 6ms/step - accuracy: 0.3334 - loss: 1.8356 - val_accuracy: 0.3187
- val_loss: 1.8665
Epoch 7/10
625/625 - 5s - 7ms/step - accuracy: 0.3346 - loss: 1.8285 - val_accuracy: 0.3320
- val_loss: 1.8433
Epoch 8/10
625/625 - 5s - 7ms/step - accuracy: 0.3390 - loss: 1.8243 - val_accuracy: 0.3417
- val_loss: 1.8367
Epoch 9/10
625/625 - 3s - 5ms/step - accuracy: 0.3401 - loss: 1.8200 - val_accuracy: 0.3353
- val_loss: 1.8498
```

```
Epoch 10/10
625/625 - 3s - 4ms/step - accuracy: 0.3415 - loss: 1.8133 - val_accuracy: 0.3386
- val_loss: 1.8288
```

[9]:
```
test_loss, test_acc = model2.evaluate(X_test, y_test, verbose=2)
print(f'\nTest accuracy for 1 hidden layer: {test_acc:.3f}')
```

```
313/313 - 0s - 1ms/step - accuracy: 0.3374 - loss: 1.8132

Test accuracy for 1 hidden layer: 0.337
```

5 Conclusion

As we decreases the the hidden layer, the test accuracy decreases due to limited learning capacity, underfitting and loss of hierarchial feature extraction.

CIFAR10 Dataset gives less accuracy than FashionMNIST Dataset, which means that CIFAR10 is more complex than FashionMNIST. CIFAR10 requires more no. of CNN Blocks to reach the accuracy of that of the FashionMNIST.

02-cifar10-cnn

September 6, 2024

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, classification_report

import keras
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from tensorflow.keras import regularizers, optimizers
from tensorflow.keras.layers import Input, Conv2D, Dense, Flatten, Dropout
from tensorflow.keras.layers import GlobalMaxPooling2D, MaxPooling2D
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Model
```

1 Loading CIFAR10 DataSet for CNN Study

[2]:
```
from tensorflow.keras.datasets import cifar10
(X_train, Y_train), (X_test, Y_test) = cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071             2s
0us/step
```

[3]:
```
X_train = X_train/255
X_test = X_test/255
Y_train_en = to_categorical(Y_train,10)
Y_test_en = to_categorical(Y_test,10)
```

2 3 Block CNN Study

[4]:
```
model = Sequential()
model.add(Conv2D(32,(4,4),input_shape = (32,32,3),activation='relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Conv2D(32,(4,4),input_shape = (32,32,3),activation='relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
```

```
model.add(Conv2D(32,(4,4),input_shape = (32,32,3),activation='relu'))
model.add(MaxPooling2D(pool_size = (2,2)))

model.add(Flatten())
model.add(Dense(128, activation ='relu'))
model.add(Dense(10, activation ='softmax'))
model.compile(loss ='categorical_crossentropy', optimizer ='adam',
metrics =['accuracy'])

model.summary()
history = model.fit(X_train, Y_train_en, epochs = 10, verbose=1,
                    validation_data=(X_test,Y_test_en))
```

```
/usr/local/lib/python3.10/dist-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 29, 29, 32)	1,568
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 32)	16,416
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 32)	0
conv2d_2 (Conv2D)	(None, 2, 2, 32)	16,416
max_pooling2d_2 (MaxPooling2D)	(None, 1, 1, 32)	0
flatten (Flatten)	(None, 32)	0

```
 dense (Dense)                        (None, 128)                 ␣
↪4,224

 dense_1 (Dense)                      (None, 10)                  ␣
↪1,290


 Total params: 39,914 (155.91 KB)

 Trainable params: 39,914 (155.91 KB)

 Non-trainable params: 0 (0.00 B)

Epoch 1/10
1563/1563             68s 42ms/step -
accuracy: 0.3132 - loss: 1.8318 - val_accuracy: 0.5151 - val_loss: 1.3419
Epoch 2/10
1563/1563             63s 40ms/step -
accuracy: 0.5172 - loss: 1.3372 - val_accuracy: 0.5497 - val_loss: 1.2618
Epoch 3/10
1563/1563             82s 40ms/step -
accuracy: 0.5743 - loss: 1.1963 - val_accuracy: 0.5842 - val_loss: 1.1662
Epoch 4/10
1563/1563             84s 41ms/step -
accuracy: 0.6081 - loss: 1.1027 - val_accuracy: 0.5873 - val_loss: 1.1687
Epoch 5/10
1563/1563             81s 41ms/step -
accuracy: 0.6371 - loss: 1.0317 - val_accuracy: 0.6256 - val_loss: 1.0631
Epoch 6/10
1563/1563             84s 42ms/step -
accuracy: 0.6553 - loss: 0.9804 - val_accuracy: 0.6397 - val_loss: 1.0304
Epoch 7/10
1563/1563             79s 40ms/step -
accuracy: 0.6691 - loss: 0.9407 - val_accuracy: 0.6416 - val_loss: 1.0386
Epoch 8/10
1563/1563             83s 41ms/step -
accuracy: 0.6824 - loss: 0.9031 - val_accuracy: 0.6287 - val_loss: 1.0610
Epoch 9/10
1563/1563             83s 42ms/step -
accuracy: 0.6987 - loss: 0.8592 - val_accuracy: 0.6595 - val_loss: 0.9947
Epoch 10/10
1563/1563             82s 42ms/step -
accuracy: 0.7021 - loss: 0.8377 - val_accuracy: 0.6596 - val_loss: 0.9881
```

[5]:
```
test_loss, test_acc = model.evaluate(X_test, Y_test_en, verbose=2)
print(f'\nTest Accuracy 3 Block Study: {test_acc:.3f}')
```

```
313/313 - 3s - 10ms/step - accuracy: 0.6596 - loss: 0.9881

Test Accuracy 3 Block Study: 0.660
```

3 2 Blocks CNN Study

[6]:
```
model1 = Sequential()
model1.add(Conv2D(32,(4,4),input_shape = (32,32,3),activation='relu'))
model1.add(MaxPooling2D(pool_size = (2,2)))
model1.add(Conv2D(32,(4,4),input_shape = (32,32,3),activation='relu'))
model1.add(MaxPooling2D(pool_size = (2,2)))

model1.add(Flatten())
model1.add(Dense(128, activation ='relu'))
model1.add(Dense(10, activation ='softmax'))
model1.compile(loss ='categorical_crossentropy', optimizer ='adam',
metrics =['accuracy'])

model1.summary()
history = model1.fit(X_train, Y_train_en, epochs = 10, verbose=1,
                     validation_data=(X_test,Y_test_en))
```

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 29, 29, 32)	1,568
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_4 (Conv2D)	(None, 11, 11, 32)	16,416
max_pooling2d_4 (MaxPooling2D)	(None, 5, 5, 32)	0
flatten_1 (Flatten)	(None, 800)	0

```
 dense_2 (Dense)                      (None, 128)                 ␣
↪102,528

 dense_3 (Dense)                      (None, 10)                  ␣
↪1,290


 Total params: 121,802 (475.79 KB)

 Trainable params: 121,802 (475.79 KB)

 Non-trainable params: 0 (0.00 B)

Epoch 1/10
1563/1563            71s 44ms/step -
accuracy: 0.3812 - loss: 1.6963 - val_accuracy: 0.5346 - val_loss: 1.2978
Epoch 2/10
1563/1563            78s 42ms/step -
accuracy: 0.5625 - loss: 1.2307 - val_accuracy: 0.5919 - val_loss: 1.1364
Epoch 3/10
1563/1563            83s 43ms/step -
accuracy: 0.6260 - loss: 1.0706 - val_accuracy: 0.6197 - val_loss: 1.0800
Epoch 4/10
1563/1563            83s 43ms/step -
accuracy: 0.6563 - loss: 0.9803 - val_accuracy: 0.6440 - val_loss: 1.0192
Epoch 5/10
1563/1563            79s 41ms/step -
accuracy: 0.6848 - loss: 0.9058 - val_accuracy: 0.6553 - val_loss: 0.9969
Epoch 6/10
1563/1563            81s 40ms/step -
accuracy: 0.7089 - loss: 0.8364 - val_accuracy: 0.6530 - val_loss: 1.0215
Epoch 7/10
1563/1563            67s 43ms/step -
accuracy: 0.7305 - loss: 0.7706 - val_accuracy: 0.6726 - val_loss: 0.9719
Epoch 8/10
1563/1563            82s 43ms/step -
accuracy: 0.7498 - loss: 0.7188 - val_accuracy: 0.6767 - val_loss: 0.9672
Epoch 9/10
1563/1563            63s 40ms/step -
accuracy: 0.7664 - loss: 0.6682 - val_accuracy: 0.6817 - val_loss: 0.9717
Epoch 10/10
1563/1563            67s 43ms/step -
accuracy: 0.7791 - loss: 0.6231 - val_accuracy: 0.6703 - val_loss: 1.0324
```

[7]:
```
test_loss, test_acc = model1.evaluate(X_test, Y_test_en, verbose=2)
print(f'\nTest accuracy 2 Block Study: {test_acc:.3f}')
```

```
313/313 - 3s - 10ms/step - accuracy: 0.6596 - loss: 0.9881

Test accuracy 2 Block Study: 0.660
```

4 1 Block CNN Study

[9]:
```
model2 = Sequential()
model2.add(Conv2D(32,(4,4),input_shape = (32,32,3),activation='relu'))
model2.add(MaxPooling2D(pool_size = (2,2)))

model2.add(Flatten())
model2.add(Dense(128, activation ='relu'))
model2.add(Dense(10, activation ='softmax'))
model2.compile(loss ='categorical_crossentropy', optimizer ='adam',
metrics =['accuracy'])

model2.summary()
history = model2.fit(X_train, Y_train_en, epochs = 10,
verbose=1,validation_data=(X_test,Y_test_en))
```

```
/usr/local/lib/python3.10/dist-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 29, 29, 32)	1,568
max_pooling2d_6 (MaxPooling2D)	(None, 14, 14, 32)	0
flatten_3 (Flatten)	(None, 6272)	0
dense_6 (Dense)	(None, 128)	802,944

```
 dense_7 (Dense)                      (None, 10)                  ␣
↪1,290


 Total params: 805,802 (3.07 MB)


 Trainable params: 805,802 (3.07 MB)


 Non-trainable params: 0 (0.00 B)


Epoch 1/10
1563/1563             57s 36ms/step -
accuracy: 0.4033 - loss: 1.6532 - val_accuracy: 0.5592 - val_loss: 1.2493
Epoch 2/10
1563/1563             81s 35ms/step -
accuracy: 0.5867 - loss: 1.1764 - val_accuracy: 0.6082 - val_loss: 1.1088
Epoch 3/10
1563/1563             82s 35ms/step -
accuracy: 0.6439 - loss: 1.0229 - val_accuracy: 0.6273 - val_loss: 1.0552
Epoch 4/10
1563/1563             56s 36ms/step -
accuracy: 0.6751 - loss: 0.9319 - val_accuracy: 0.6416 - val_loss: 1.0583
Epoch 5/10
1563/1563             82s 35ms/step -
accuracy: 0.7084 - loss: 0.8370 - val_accuracy: 0.6358 - val_loss: 1.0507
Epoch 6/10
1563/1563             82s 36ms/step -
accuracy: 0.7390 - loss: 0.7568 - val_accuracy: 0.6433 - val_loss: 1.0747
Epoch 7/10
1563/1563             82s 36ms/step -
accuracy: 0.7560 - loss: 0.7044 - val_accuracy: 0.6329 - val_loss: 1.0941
Epoch 8/10
1563/1563             82s 35ms/step -
accuracy: 0.7787 - loss: 0.6370 - val_accuracy: 0.6370 - val_loss: 1.1502
Epoch 9/10
1563/1563             60s 38ms/step -
accuracy: 0.7974 - loss: 0.5741 - val_accuracy: 0.6451 - val_loss: 1.1211
Epoch 10/10
1563/1563             78s 36ms/step -
accuracy: 0.8219 - loss: 0.5147 - val_accuracy: 0.6463 - val_loss: 1.1989
```

[10]:
```
test_loss, test_acc = model2.evaluate(X_test, Y_test_en, verbose=2)
print(f'\nTest accuracy for 1 Block Study: {test_acc:.3f}')
```

```
313/313 - 2s - 8ms/step - accuracy: 0.6463 - loss: 1.1989
```

```
Test accuracy for 1 Block Study: 0.646
```

5 Conclusion (CIFAR 10 CNN Study):

If your problem requires detailed feature extraction, decreasing the number of blocks too much may decrease the performance.

In 2 Block and 3 Block CNN Study, accuracy (0.66) didn't changed but in 1 Block CNN, it got decreased to (0.646).

CIFAR10 shows lesser accuracy than FashionMNIST at the same no. of hidden layers, which means CIFAR10 is more complex dataset than FashionMNIST. CIFAR 10 requires more no. of hidden layers to reach the accuracy level of that of the FashionMNIST.

03-fashionmnist-ann

September 6, 2024

[1]:
```
import matplotlib.pyplot as plt
import numpy as np

from tensorflow import keras
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from keras.models import Sequential
from keras.utils import to_categorical
```

1 Loading FashionMNIST Dataset

[2]:
```
fashion_mnist = keras.datasets.fashion_mnist
(X_train,y_train),(X_test,y_test) = fashion_mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-labels-idx1-ubyte.gz
29515/29515             0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-images-idx3-ubyte.gz
26421880/26421880             2s
0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/t10k-labels-idx1-ubyte.gz
5148/5148             0s 1us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/t10k-images-idx3-ubyte.gz
4422102/4422102             1s
0us/step
```

[3]:
```
X_train=X_train/255.0
X_test=X_test/255.0
n=len(np.unique(y_train))
```

[4]:
```
y_train=to_categorical(y_train,n)
y_test=to_categorical(y_test,n)
```

2 2 Hidden Layer ANN

[5]:
```
model = Sequential()
model.add(Flatten(input_shape=(28, 28)))
model.add(Dense(units=32,activation='relu'))
model.add(Dense(units=64,activation='relu'))
model.add(Dense(units=10,activation='softmax'))
model.compile(optimizer='adam',loss='categorical_crossentropy',
              metrics=['accuracy'])
answer=model.fit(X_train,y_train,epochs=10,verbose=2,
                 validation_split=0.2,batch_size=64)
```

```
/usr/local/lib/python3.10/dist-
packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
Epoch 1/10
750/750 - 4s - 6ms/step - accuracy: 0.7916 - loss: 0.5993 - val_accuracy: 0.8402
- val_loss: 0.4618
Epoch 2/10
750/750 - 2s - 2ms/step - accuracy: 0.8513 - loss: 0.4201 - val_accuracy: 0.8531
- val_loss: 0.4127
Epoch 3/10
750/750 - 3s - 3ms/step - accuracy: 0.8632 - loss: 0.3787 - val_accuracy: 0.8610
- val_loss: 0.3865
Epoch 4/10
750/750 - 2s - 3ms/step - accuracy: 0.8719 - loss: 0.3561 - val_accuracy: 0.8702
- val_loss: 0.3660
Epoch 5/10
750/750 - 3s - 5ms/step - accuracy: 0.8784 - loss: 0.3360 - val_accuracy: 0.8734
- val_loss: 0.3532
Epoch 6/10
750/750 - 2s - 3ms/step - accuracy: 0.8818 - loss: 0.3216 - val_accuracy: 0.8703
- val_loss: 0.3602
Epoch 7/10
750/750 - 2s - 2ms/step - accuracy: 0.8853 - loss: 0.3117 - val_accuracy: 0.8725
- val_loss: 0.3596
Epoch 8/10
750/750 - 2s - 3ms/step - accuracy: 0.8866 - loss: 0.3055 - val_accuracy: 0.8716
- val_loss: 0.3533
Epoch 9/10
750/750 - 2s - 3ms/step - accuracy: 0.8925 - loss: 0.2918 - val_accuracy: 0.8780
- val_loss: 0.3371
Epoch 10/10
750/750 - 2s - 2ms/step - accuracy: 0.8942 - loss: 0.2860 - val_accuracy: 0.8767
- val_loss: 0.3536
```

[6]:
```
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
print(f'\nTest accuracy for 2 Hidden Layers: {test_acc:.3f}')
```

```
313/313 - 1s - 2ms/step - accuracy: 0.8677 - loss: 0.3839

Test accuracy for 2 Hidden Layers: 0.868
```

3 3 hidden layer ANN

[7]:
```
model1 = Sequential()
model1.add(Flatten(input_shape=(28, 28)))
model1.add(Dense(units=32,activation='relu'))
model1.add(Dense(units=64,activation='relu'))
model1.add(Dense(units=64,activation='relu'))
model1.add(Dense(units=10,activation='softmax'))
model1.compile(optimizer='adam',loss='categorical_crossentropy',
               metrics=['accuracy'])
answer=model1.fit(X_train,y_train,epochs=10,verbose=2,
                  validation_split=0.2,batch_size=64)
```

```
Epoch 1/10
750/750 - 4s - 5ms/step - accuracy: 0.7864 - loss: 0.6090 - val_accuracy: 0.8408
- val_loss: 0.4511
Epoch 2/10
750/750 - 2s - 3ms/step - accuracy: 0.8515 - loss: 0.4148 - val_accuracy: 0.8538
- val_loss: 0.4029
Epoch 3/10
750/750 - 3s - 3ms/step - accuracy: 0.8643 - loss: 0.3739 - val_accuracy: 0.8478
- val_loss: 0.4099
Epoch 4/10
750/750 - 2s - 3ms/step - accuracy: 0.8718 - loss: 0.3508 - val_accuracy: 0.8656
- val_loss: 0.3736
Epoch 5/10
750/750 - 3s - 4ms/step - accuracy: 0.8786 - loss: 0.3328 - val_accuracy: 0.8707
- val_loss: 0.3574
Epoch 6/10
750/750 - 2s - 2ms/step - accuracy: 0.8813 - loss: 0.3189 - val_accuracy: 0.8728
- val_loss: 0.3513
Epoch 7/10
750/750 - 3s - 4ms/step - accuracy: 0.8864 - loss: 0.3097 - val_accuracy: 0.8758
- val_loss: 0.3435
Epoch 8/10
750/750 - 2s - 2ms/step - accuracy: 0.8900 - loss: 0.2977 - val_accuracy: 0.8732
- val_loss: 0.3496
Epoch 9/10
750/750 - 2s - 2ms/step - accuracy: 0.8932 - loss: 0.2872 - val_accuracy: 0.8798
- val_loss: 0.3336
```

```
Epoch 10/10
750/750 - 3s - 4ms/step - accuracy: 0.8958 - loss: 0.2807 - val_accuracy: 0.8731
- val_loss: 0.3494
```

[8]:
```
test_loss, test_acc = model1.evaluate(X_test, y_test, verbose=2)
print(f'\nTest accuracy for 3 Hidden Layers: {test_acc:.3f}')
```

```
313/313 - 1s - 2ms/step - accuracy: 0.8653 - loss: 0.3724

Test accuracy for 3 Hidden Layers: 0.865
```

4 1 Hidden Layer ANN

[9]:
```
model2 = Sequential()
model2.add(Flatten(input_shape=(28,28)))
model2.add(Dense(units=32,activation='relu'))
model2.add(Dense(units=10,activation='softmax'))
model2.compile(optimizer='adam',loss='categorical_crossentropy',
               metrics=['accuracy'])
answer=model2.fit(X_train,y_train,epochs=10,verbose=2,
                  validation_split=0.2,batch_size=64)
```

```
Epoch 1/10
750/750 - 3s - 3ms/step - accuracy: 0.7848 - loss: 0.6347 - val_accuracy: 0.8336
- val_loss: 0.4684
Epoch 2/10
750/750 - 2s - 3ms/step - accuracy: 0.8467 - loss: 0.4413 - val_accuracy: 0.8467
- val_loss: 0.4430
Epoch 3/10
750/750 - 1s - 2ms/step - accuracy: 0.8571 - loss: 0.4067 - val_accuracy: 0.8583
- val_loss: 0.4096
Epoch 4/10
750/750 - 2s - 3ms/step - accuracy: 0.8646 - loss: 0.3852 - val_accuracy: 0.8577
- val_loss: 0.4093
Epoch 5/10
750/750 - 4s - 5ms/step - accuracy: 0.8689 - loss: 0.3691 - val_accuracy: 0.8608
- val_loss: 0.3904
Epoch 6/10
750/750 - 2s - 2ms/step - accuracy: 0.8715 - loss: 0.3587 - val_accuracy: 0.8652
- val_loss: 0.3805
Epoch 7/10
750/750 - 3s - 3ms/step - accuracy: 0.8769 - loss: 0.3469 - val_accuracy: 0.8703
- val_loss: 0.3689
Epoch 8/10
750/750 - 2s - 3ms/step - accuracy: 0.8786 - loss: 0.3394 - val_accuracy: 0.8634
- val_loss: 0.3815
Epoch 9/10
```

```
750/750 - 3s - 4ms/step - accuracy: 0.8808 - loss: 0.3317 - val_accuracy: 0.8627
- val_loss: 0.3890
Epoch 10/10
750/750 - 4s - 5ms/step - accuracy: 0.8828 - loss: 0.3254 - val_accuracy: 0.8652
- val_loss: 0.3700
```

[11]:
```
test_loss, test_acc = model2.evaluate(X_test, y_test, verbose=2)
print(f'\nTest accuracy for 1 Hidden Layer: {test_acc:.3f}')
```

```
313/313 - 0s - 1ms/step - accuracy: 0.8610 - loss: 0.3981

Test accuracy for 1 Hidden Layer: 0.861
```

5 Conclusion

As the FashionMNIST Dataset is simpler than CIFAR10 Dataset, it shows more test accuracy at the same no. of hidden layer than that in CIFAR10.

For 2 hidden layers it was the highest means than that of 1 and 3 hidden layer architectures. Test Accuracy was lower for 1 hidden layer architecture than that of 3 layer architecture.

As we decrease the hidden layer, test accuracy decreases (at 1 hidden layer architecture) due to limited learning capacity, underfitting and loss of hierarchial feature extraction. It also suffers overfitting (at 3 hidden layer architecture) which lead to decrement in test accuracy.

04-fashionmnist-cnn

September 6, 2024

[1]:
```
from sklearn.metrics import classification_report
import numpy as np
import matplotlib.pyplot as plt

import keras
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from tensorflow.keras import regularizers, optimizers
from tensorflow.keras.layers import Input, Conv2D, Dense, Flatten, Dropout
from tensorflow.keras.layers import GlobalMaxPooling2D, MaxPooling2D
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Model
```

1 Loading FashionMNIST Dataset

[2]:
```
(x_train, y_train), (x_test, y_test) =keras.datasets.fashion_mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-labels-idx1-ubyte.gz
29515/29515             0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-images-idx3-ubyte.gz
26421880/26421880             0s
0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/t10k-labels-idx1-ubyte.gz
5148/5148             0s 1us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/t10k-images-idx3-ubyte.gz
4422102/4422102             0s
0us/step
```

[3]:
```
x_train = x_train / 255.0
x_test = x_test / 255.0
```

2 3 Block CNN

[4]:
```
model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(3, 3), strides=(1, 1),
                 padding='valid', activation='relu',
                 input_shape=(28, 28, 1)))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Dropout(rate=0.25))

model.add(Conv2D(filters=32, kernel_size=(3, 3),strides=(1, 1),
                                padding='valid', activation='relu',
                                input_shape=(28, 28, 1)))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Dropout(rate=0.25))

model.add(Conv2D(filters=32, kernel_size=(3, 3),strides=(1, 1),
                 padding='valid', activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Dropout(rate=0.25))

model.add(Flatten())
model.add(Dense(units=128, activation='relu'))
model.add(Dense(units=10, activation='softmax'))
model.compile(loss=keras.losses.sparse_categorical_crossentropy,
              optimizer=keras.optimizers.Adam(), metrics=['accuracy'])
model.summary()
```

```
/usr/local/lib/python3.10/dist-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0

```
dropout (Dropout)                    (None, 13, 13, 32)                ␣
↪ 0

conv2d_1 (Conv2D)                    (None, 11, 11, 32)                ␣
↪9,248

max_pooling2d_1 (MaxPooling2D)       (None, 5, 5, 32)                  ␣
↪ 0

dropout_1 (Dropout)                  (None, 5, 5, 32)                  ␣
↪ 0

conv2d_2 (Conv2D)                    (None, 3, 3, 32)                  ␣
↪9,248

max_pooling2d_2 (MaxPooling2D)       (None, 1, 1, 32)                  ␣
↪ 0

dropout_2 (Dropout)                  (None, 1, 1, 32)                  ␣
↪ 0

flatten (Flatten)                    (None, 32)                        ␣
↪ 0

dense (Dense)                        (None, 128)                       ␣
↪4,224

dense_1 (Dense)                      (None, 10)                        ␣
↪1,290


Total params: 24,330 (95.04 KB)

Trainable params: 24,330 (95.04 KB)

Non-trainable params: 0 (0.00 B)
```

[5]:
```
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
history = model.fit(x_train, y_train, batch_size=256, epochs=10,
                    validation_split=0.2, verbose=1)
```

```
Epoch 1/10
188/188             42s 209ms/step -
```

```
accuracy: 0.3675 - loss: 1.6517 - val_accuracy: 0.7321 - val_loss: 0.7728
Epoch 2/10
188/188             39s 209ms/step -
accuracy: 0.6906 - loss: 0.8311 - val_accuracy: 0.7638 - val_loss: 0.6302
Epoch 3/10
188/188             41s 210ms/step -
accuracy: 0.7338 - loss: 0.7217 - val_accuracy: 0.7922 - val_loss: 0.5814
Epoch 4/10
188/188             36s 193ms/step -
accuracy: 0.7548 - loss: 0.6669 - val_accuracy: 0.8026 - val_loss: 0.5480
Epoch 5/10
188/188             43s 203ms/step -
accuracy: 0.7693 - loss: 0.6294 - val_accuracy: 0.8145 - val_loss: 0.5170
Epoch 6/10
188/188             40s 199ms/step -
accuracy: 0.7833 - loss: 0.5985 - val_accuracy: 0.8163 - val_loss: 0.5087
Epoch 7/10
188/188             39s 191ms/step -
accuracy: 0.7852 - loss: 0.5799 - val_accuracy: 0.8285 - val_loss: 0.4807
Epoch 8/10
188/188             37s 199ms/step -
accuracy: 0.7963 - loss: 0.5527 - val_accuracy: 0.8300 - val_loss: 0.4736
Epoch 9/10
188/188             40s 211ms/step -
accuracy: 0.8082 - loss: 0.5326 - val_accuracy: 0.8344 - val_loss: 0.4570
Epoch 10/10
188/188             36s 193ms/step -
accuracy: 0.8103 - loss: 0.5216 - val_accuracy: 0.8446 - val_loss: 0.4335
```

[6]:
```
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f'\nTest accuracy for 3 block CNN: {test_acc:.3f}')
```

```
313/313 - 2s - 8ms/step - accuracy: 0.8442 - loss: 0.4446

Test accuracy for 3 block CNN: 0.844
```

3 2 Blocks CNN

[7]:
```
model1 = Sequential()
model1.add(Conv2D(filters=32, kernel_size=(3, 3),strides=(1, 1),
                  padding='valid',activation='relu', input_shape=(28, 28, 1)))
model1.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model1.add(Dropout(rate=0.25))

model1.add(Conv2D(filters=32, kernel_size=(3, 3), strides=(1, 1),
                  padding='valid',activation='relu', input_shape=(28, 28, 1)))
model1.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
```

```
model1.add(Dropout(rate=0.25))

model1.add(Flatten())
model1.add(Dense(units=128, activation='relu'))
model1.add(Dense(units=10, activation='softmax'))
model1.compile(loss=keras.losses.sparse_categorical_crossentropy,
               optimizer=keras.optimizers.Adam(),
               metrics=['accuracy'])
model1.summary()
```

```
Model: "sequential_1"
```

```
 Layer (type)                        Output Shape                      ␣
↪Param #

 conv2d_3 (Conv2D)                   (None, 26, 26, 32)                ␣
↪320

 max_pooling2d_3 (MaxPooling2D)      (None, 13, 13, 32)                ␣
↪  0

 dropout_3 (Dropout)                 (None, 13, 13, 32)                ␣
↪  0

 conv2d_4 (Conv2D)                   (None, 11, 11, 32)                ␣
↪9,248

 max_pooling2d_4 (MaxPooling2D)      (None, 5, 5, 32)                  ␣
↪  0

 dropout_4 (Dropout)                 (None, 5, 5, 32)                  ␣
↪  0

 flatten_1 (Flatten)                 (None, 800)                       ␣
↪  0

 dense_2 (Dense)                     (None, 128)                       ␣
↪102,528

 dense_3 (Dense)                     (None, 10)                        ␣
↪1,290


 Total params: 113,386 (442.91 KB)
```

Trainable params: 113,386 (442.91 KB)

Non-trainable params: 0 (0.00 B)

[8]:
```
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
```

[9]:
```
history = model1.fit(x_train, y_train, batch_size=256, epochs=10,
                     validation_split=0.2, verbose=1)
```

```
Epoch 1/10
188/188             41s 203ms/step -
accuracy: 0.5993 - loss: 1.1462 - val_accuracy: 0.8108 - val_loss: 0.5147
Epoch 2/10
188/188             41s 217ms/step -
accuracy: 0.8079 - loss: 0.5148 - val_accuracy: 0.8413 - val_loss: 0.4284
Epoch 3/10
188/188             39s 204ms/step -
accuracy: 0.8383 - loss: 0.4484 - val_accuracy: 0.8625 - val_loss: 0.3942
Epoch 4/10
188/188             39s 192ms/step -
accuracy: 0.8504 - loss: 0.4149 - val_accuracy: 0.8709 - val_loss: 0.3641
Epoch 5/10
188/188             37s 194ms/step -
accuracy: 0.8595 - loss: 0.3807 - val_accuracy: 0.8800 - val_loss: 0.3380
Epoch 6/10
188/188             39s 210ms/step -
accuracy: 0.8704 - loss: 0.3583 - val_accuracy: 0.8799 - val_loss: 0.3299
Epoch 7/10
188/188             35s 188ms/step -
accuracy: 0.8722 - loss: 0.3475 - val_accuracy: 0.8856 - val_loss: 0.3178
Epoch 8/10
188/188             39s 178ms/step -
accuracy: 0.8815 - loss: 0.3256 - val_accuracy: 0.8891 - val_loss: 0.3112
Epoch 9/10
188/188             41s 178ms/step -
accuracy: 0.8847 - loss: 0.3159 - val_accuracy: 0.8915 - val_loss: 0.3011
Epoch 10/10
188/188             42s 183ms/step -
accuracy: 0.8839 - loss: 0.3132 - val_accuracy: 0.8953 - val_loss: 0.2880
```

[10]:
```
test_loss, test_acc = model1.evaluate(x_test, y_test, verbose=2)
print(f'\nTest accuracy for 2 Block CNN: {test_acc:.3f}')
```

```
313/313 - 2s - 6ms/step - accuracy: 0.8929 - loss: 0.3017
```

```
Test accuracy for 2 block CNN: 0.893
```

4 Conclusion

As Blocks were decreased from 3 to 2, it gave more accuracy. In block 3, it might have lead to overfitting and FashionMNIST Data may be simpler.

FashionMNIST give more accuracy at same no. of blocks than that of CIFAR10 Dataset.