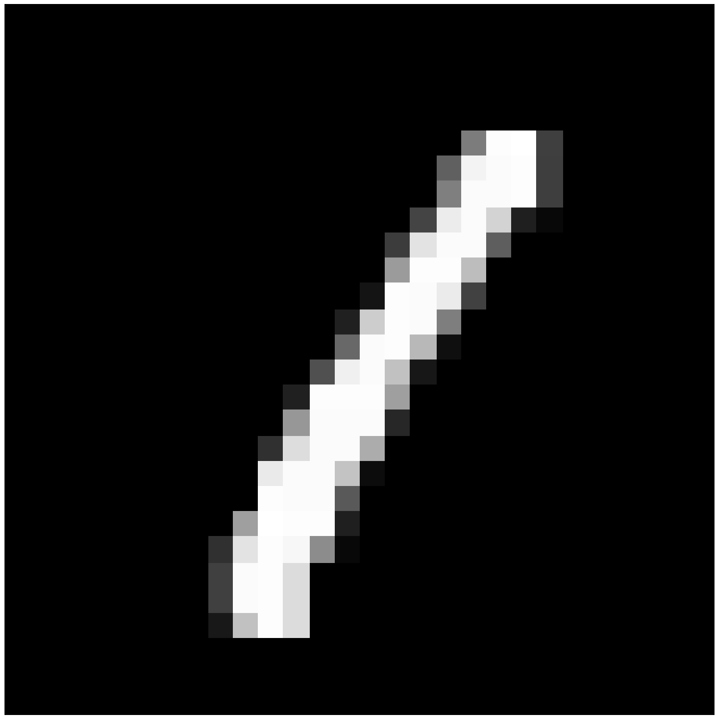
DA_2

September 1, 2025

Name:	Tufan Kundu
Registration no:	24MDT0184
Course Name:	Deep Learning Lab
Course Code:	PMDS603P
Digital Assessment:	2

0.1 Question1. Today, we will try to recall the work done in the previous lab first. The second problem attempted in the last lab was to use MNIST dataset which contains handwritten numbers (their images) from 0 to 9 digits. First try to fit a simple neural network model. Let us import the necessary modules required for this along with the dataset. It contains 70000 handwritten images of digits from 0 to 9. So its a 10 class classification problem. Lets try to create a model that can do the classification task.

[4]:
```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense,Dropout,Flatten
from keras.optimizers import SGD
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
batch_size = 128
num_classes = 10
epochs = 50
(x_train,y_train), (x_test,y_test) = mnist.load_data()
plt.imshow(x_train[3],cmap='gray')
plt.axis('off')
plt.show()
```



[5]:
```
x_train = x_train.reshape(60000,784)
x_test = x_test.reshape(10000,784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train/=255
x_test/=255
print(x_train.shape[0],'train samples')
print(x_test.shape[0],'test samples')
y_train = keras.utils.to_categorical(y_train,num_classes)
y_test_ = keras.utils.to_categorical(y_test,num_classes)
```

```
60000 train samples
10000 test samples
```

0.1.1 Without dropout with ReLU activation

[6]:
```
model = Sequential()
model.add(Dense(512, activation = 'relu',input_shape = (784,)))
model.add(Dense(512, activation = 'relu'))
model.add(Dense(10, activation = 'softmax'))
model.summary()
sgd1 = SGD(learning_rate=0.01)
model.compile(loss = 'categorical_crossentropy', optimizer = sgd1, metrics =
    ['accuracy'])
```

```
history = model.
↪fit(x_train,y_train,batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(x_test,y
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_3 (Dense)	(None, 512)	401,920
dense_4 (Dense)	(None, 512)	262,656
dense_5 (Dense)	(None, 10)	5,130

Total params: 669,706 (2.55 MB)

Trainable params: 669,706 (2.55 MB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/50
469/469             5s 9ms/step -
accuracy: 0.6011 - loss: 1.6053 - val_accuracy: 0.8828 - val_loss: 0.5087
Epoch 2/50
469/469             3s 6ms/step -
accuracy: 0.8802 - loss: 0.4768 - val_accuracy: 0.9051 - val_loss: 0.3595
Epoch 3/50
469/469             3s 7ms/step -
accuracy: 0.8989 - loss: 0.3613 - val_accuracy: 0.9145 - val_loss: 0.3125
Epoch 4/50
469/469             3s 7ms/step -
accuracy: 0.9107 - loss: 0.3174 - val_accuracy: 0.9214 - val_loss: 0.2854
Epoch 5/50
469/469             3s 6ms/step -
accuracy: 0.9158 - loss: 0.2946 - val_accuracy: 0.9269 - val_loss: 0.2666
Epoch 6/50
469/469             3s 6ms/step -
accuracy: 0.9220 - loss: 0.2774 - val_accuracy: 0.9301 - val_loss: 0.2509
Epoch 7/50
469/469             3s 7ms/step -
accuracy: 0.9259 - loss: 0.2599 - val_accuracy: 0.9323 - val_loss: 0.2413
Epoch 8/50
469/469             3s 7ms/step -
accuracy: 0.9323 - loss: 0.2392 - val_accuracy: 0.9373 - val_loss: 0.2276
```

```
Epoch 9/50
469/469            3s 6ms/step -
accuracy: 0.9350 - loss: 0.2308 - val_accuracy: 0.9384 - val_loss: 0.2186
Epoch 10/50
469/469            3s 7ms/step -
accuracy: 0.9360 - loss: 0.2213 - val_accuracy: 0.9418 - val_loss: 0.2096
Epoch 11/50
469/469            3s 6ms/step -
accuracy: 0.9404 - loss: 0.2095 - val_accuracy: 0.9437 - val_loss: 0.2027
Epoch 12/50
469/469            3s 6ms/step -
accuracy: 0.9427 - loss: 0.2047 - val_accuracy: 0.9450 - val_loss: 0.1952
Epoch 13/50
469/469            3s 6ms/step -
accuracy: 0.9445 - loss: 0.1950 - val_accuracy: 0.9469 - val_loss: 0.1887
Epoch 14/50
469/469            3s 7ms/step -
accuracy: 0.9462 - loss: 0.1879 - val_accuracy: 0.9481 - val_loss: 0.1813
Epoch 15/50
469/469            3s 7ms/step -
accuracy: 0.9484 - loss: 0.1783 - val_accuracy: 0.9499 - val_loss: 0.1750
Epoch 16/50
469/469            3s 7ms/step -
accuracy: 0.9511 - loss: 0.1703 - val_accuracy: 0.9510 - val_loss: 0.1699
Epoch 17/50
469/469            3s 7ms/step -
accuracy: 0.9543 - loss: 0.1621 - val_accuracy: 0.9514 - val_loss: 0.1651
Epoch 18/50
469/469            3s 7ms/step -
accuracy: 0.9543 - loss: 0.1602 - val_accuracy: 0.9532 - val_loss: 0.1604
Epoch 19/50
469/469            3s 6ms/step -
accuracy: 0.9560 - loss: 0.1575 - val_accuracy: 0.9554 - val_loss: 0.1554
Epoch 20/50
469/469            3s 6ms/step -
accuracy: 0.9575 - loss: 0.1505 - val_accuracy: 0.9546 - val_loss: 0.1531
Epoch 21/50
469/469            3s 6ms/step -
accuracy: 0.9602 - loss: 0.1442 - val_accuracy: 0.9573 - val_loss: 0.1477
Epoch 22/50
469/469            3s 7ms/step -
accuracy: 0.9605 - loss: 0.1399 - val_accuracy: 0.9573 - val_loss: 0.1448
Epoch 23/50
469/469            3s 7ms/step -
accuracy: 0.9627 - loss: 0.1343 - val_accuracy: 0.9588 - val_loss: 0.1402
Epoch 24/50
469/469            3s 6ms/step -
accuracy: 0.9623 - loss: 0.1338 - val_accuracy: 0.9608 - val_loss: 0.1368
```

```
Epoch 25/50
469/469            3s 6ms/step -
accuracy: 0.9643 - loss: 0.1273 - val_accuracy: 0.9612 - val_loss: 0.1336
Epoch 26/50
469/469            3s 7ms/step -
accuracy: 0.9655 - loss: 0.1246 - val_accuracy: 0.9610 - val_loss: 0.1322
Epoch 27/50
469/469            4s 8ms/step -
accuracy: 0.9650 - loss: 0.1225 - val_accuracy: 0.9625 - val_loss: 0.1279
Epoch 28/50
469/469            4s 7ms/step -
accuracy: 0.9673 - loss: 0.1169 - val_accuracy: 0.9632 - val_loss: 0.1269
Epoch 29/50
469/469            3s 6ms/step -
accuracy: 0.9692 - loss: 0.1129 - val_accuracy: 0.9638 - val_loss: 0.1238
Epoch 30/50
469/469            3s 6ms/step -
accuracy: 0.9683 - loss: 0.1120 - val_accuracy: 0.9645 - val_loss: 0.1216
Epoch 31/50
469/469            3s 7ms/step -
accuracy: 0.9695 - loss: 0.1093 - val_accuracy: 0.9649 - val_loss: 0.1190
Epoch 32/50
469/469            3s 7ms/step -
accuracy: 0.9714 - loss: 0.1053 - val_accuracy: 0.9656 - val_loss: 0.1170
Epoch 33/50
469/469            4s 7ms/step -
accuracy: 0.9704 - loss: 0.1042 - val_accuracy: 0.9664 - val_loss: 0.1159
Epoch 34/50
469/469            7s 14ms/step -
accuracy: 0.9714 - loss: 0.1033 - val_accuracy: 0.9668 - val_loss: 0.1133
Epoch 35/50
469/469            4s 8ms/step -
accuracy: 0.9734 - loss: 0.0968 - val_accuracy: 0.9667 - val_loss: 0.1122
Epoch 36/50
469/469            3s 7ms/step -
accuracy: 0.9734 - loss: 0.0944 - val_accuracy: 0.9677 - val_loss: 0.1091
Epoch 37/50
469/469            3s 7ms/step -
accuracy: 0.9739 - loss: 0.0940 - val_accuracy: 0.9682 - val_loss: 0.1076
Epoch 38/50
469/469            3s 7ms/step -
accuracy: 0.9757 - loss: 0.0879 - val_accuracy: 0.9684 - val_loss: 0.1057
Epoch 39/50
469/469            3s 7ms/step -
accuracy: 0.9758 - loss: 0.0879 - val_accuracy: 0.9689 - val_loss: 0.1044
Epoch 40/50
469/469            5s 10ms/step -
accuracy: 0.9762 - loss: 0.0855 - val_accuracy: 0.9693 - val_loss: 0.1027
```

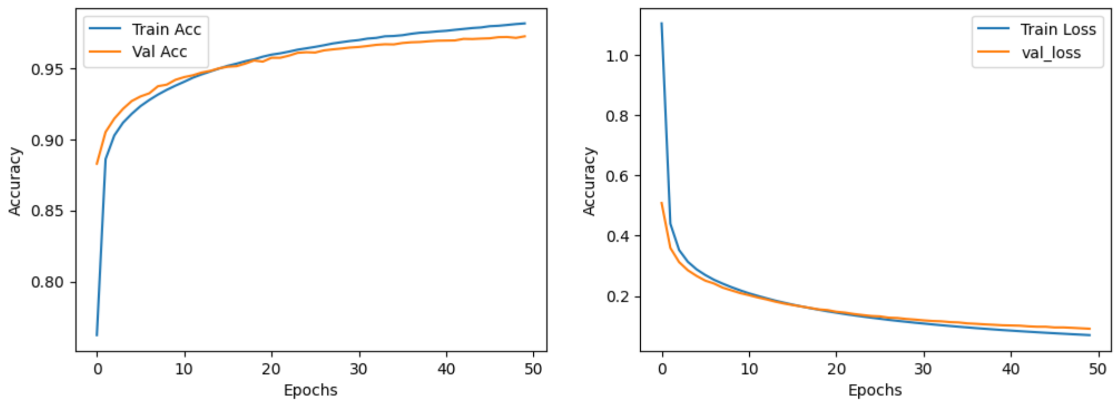
```
Epoch 41/50
469/469              4s 8ms/step -
accuracy: 0.9765 - loss: 0.0850 - val_accuracy: 0.9694 - val_loss: 0.1021
Epoch 42/50
469/469              4s 8ms/step -
accuracy: 0.9779 - loss: 0.0823 - val_accuracy: 0.9695 - val_loss: 0.1014
Epoch 43/50
469/469              5s 10ms/step -
accuracy: 0.9777 - loss: 0.0811 - val_accuracy: 0.9706 - val_loss: 0.0992
Epoch 44/50
469/469              7s 15ms/step -
accuracy: 0.9784 - loss: 0.0787 - val_accuracy: 0.9705 - val_loss: 0.0979
Epoch 45/50
469/469              8s 11ms/step -
accuracy: 0.9788 - loss: 0.0782 - val_accuracy: 0.9708 - val_loss: 0.0978
Epoch 46/50
469/469              4s 9ms/step -
accuracy: 0.9795 - loss: 0.0778 - val_accuracy: 0.9710 - val_loss: 0.0955
Epoch 47/50
469/469              5s 9ms/step -
accuracy: 0.9797 - loss: 0.0745 - val_accuracy: 0.9718 - val_loss: 0.0956
Epoch 48/50
469/469              3s 7ms/step -
accuracy: 0.9802 - loss: 0.0719 - val_accuracy: 0.9719 - val_loss: 0.0943
Epoch 49/50
469/469              4s 8ms/step -
accuracy: 0.9812 - loss: 0.0705 - val_accuracy: 0.9713 - val_loss: 0.0927
Epoch 50/50
469/469              3s 7ms/step -
accuracy: 0.9823 - loss: 0.0677 - val_accuracy: 0.9724 - val_loss: 0.0912
```

[7]:
```
score = model.evaluate(x_test,y_test_, verbose = 1)
print("Test loss:", score[0])
print(f"Test Accuracy:{score[1]*100:.2f}%")
```

```
313/313              1s 3ms/step -
accuracy: 0.9668 - loss: 0.1081
Test loss: 0.09124796092510223
Test Accuracy:97.24%
```

[8]:
```
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.subplot(1,2,2)
```

```
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label = 'val_loss')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



0.2 Without dropout using sigmoid activation

[9]:
```
model = Sequential()
model.add(Dense(512, activation = 'sigmoid',input_shape = (784,)))
model.add(Dense(512, activation = 'sigmoid'))
model.add(Dense(10, activation = 'softmax'))
model.summary()
sgd1 = SGD(learning_rate=0.01)
model.compile(loss = 'categorical_crossentropy', optimizer = sgd1, metrics =␣
 ↪['accuracy'])
history = model.
 ↪fit(x_train,y_train,batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(x_test,y
score = model.evaluate(x_test,y_test_, verbose = 1)
print("Test loss:", score[0])
print(f"Test Accuracy:{score[1]*100:.2f}%")
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
dense_6 (Dense)	(None, 512)	401,920
dense_7 (Dense)	(None, 512)	262,656

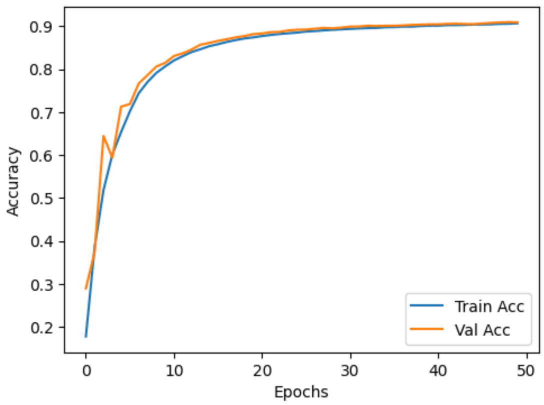
```
 dense_8 (Dense)                      (None, 10)                      5,130

 Total params: 669,706 (2.55 MB)

 Trainable params: 669,706 (2.55 MB)

 Non-trainable params: 0 (0.00 B)

Epoch 1/50
469/469              5s 9ms/step -
accuracy: 0.1307 - loss: 2.2957 - val_accuracy: 0.2895 - val_loss: 2.2310
Epoch 2/50
469/469              4s 8ms/step -
accuracy: 0.3464 - loss: 2.2137 - val_accuracy: 0.3752 - val_loss: 2.1390
Epoch 3/50
469/469              3s 7ms/step -
accuracy: 0.4897 - loss: 2.1150 - val_accuracy: 0.6444 - val_loss: 2.0076
Epoch 4/50
469/469              3s 7ms/step -
accuracy: 0.5914 - loss: 1.9729 - val_accuracy: 0.5940 - val_loss: 1.8208
Epoch 5/50
469/469              3s 7ms/step -
accuracy: 0.6405 - loss: 1.7719 - val_accuracy: 0.7126 - val_loss: 1.5753
Epoch 6/50
469/469              4s 8ms/step -
accuracy: 0.6928 - loss: 1.5271 - val_accuracy: 0.7186 - val_loss: 1.3310
Epoch 7/50
469/469              5s 11ms/step -
accuracy: 0.7367 - loss: 1.2956 - val_accuracy: 0.7665 - val_loss: 1.1272
Epoch 8/50
469/469              4s 8ms/step -
accuracy: 0.7633 - loss: 1.1038 - val_accuracy: 0.7860 - val_loss: 0.9749
Epoch 9/50
469/469              3s 7ms/step -
accuracy: 0.7829 - loss: 0.9613 - val_accuracy: 0.8059 - val_loss: 0.8597
Epoch 10/50
469/469              3s 7ms/step -
accuracy: 0.8022 - loss: 0.8531 - val_accuracy: 0.8145 - val_loss: 0.7728
Epoch 11/50
469/469              4s 8ms/step -
accuracy: 0.8178 - loss: 0.7701 - val_accuracy: 0.8305 - val_loss: 0.7045
Epoch 12/50
469/469              3s 7ms/step -
accuracy: 0.8279 - loss: 0.7078 - val_accuracy: 0.8367 - val_loss: 0.6524
Epoch 13/50
```

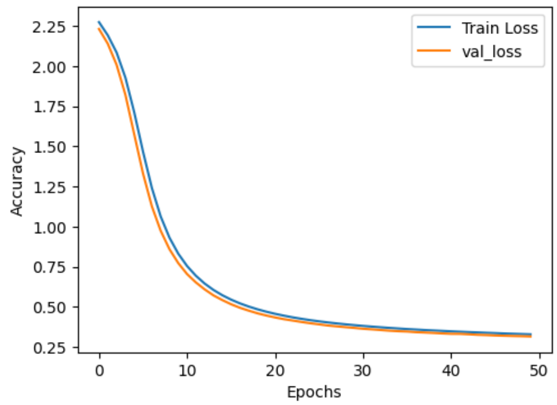
```
469/469              3s 7ms/step -
accuracy: 0.8368 - loss: 0.6597 - val_accuracy: 0.8455 - val_loss: 0.6090
Epoch 14/50
469/469              3s 7ms/step -
accuracy: 0.8461 - loss: 0.6126 - val_accuracy: 0.8567 - val_loss: 0.5716
Epoch 15/50
469/469              3s 7ms/step -
accuracy: 0.8530 - loss: 0.5838 - val_accuracy: 0.8609 - val_loss: 0.5424
Epoch 16/50
469/469              3s 7ms/step -
accuracy: 0.8572 - loss: 0.5504 - val_accuracy: 0.8656 - val_loss: 0.5164
Epoch 17/50
469/469              3s 7ms/step -
accuracy: 0.8638 - loss: 0.5206 - val_accuracy: 0.8695 - val_loss: 0.4945
Epoch 18/50
469/469              3s 6ms/step -
accuracy: 0.8632 - loss: 0.5141 - val_accuracy: 0.8738 - val_loss: 0.4764
Epoch 19/50
469/469              3s 7ms/step -
accuracy: 0.8701 - loss: 0.4913 - val_accuracy: 0.8769 - val_loss: 0.4600
Epoch 20/50
469/469              3s 7ms/step -
accuracy: 0.8711 - loss: 0.4760 - val_accuracy: 0.8813 - val_loss: 0.4462
Epoch 21/50
469/469              3s 7ms/step -
accuracy: 0.8785 - loss: 0.4564 - val_accuracy: 0.8829 - val_loss: 0.4341
Epoch 22/50
469/469              3s 7ms/step -
accuracy: 0.8791 - loss: 0.4465 - val_accuracy: 0.8860 - val_loss: 0.4233
Epoch 23/50
469/469              4s 9ms/step -
accuracy: 0.8815 - loss: 0.4374 - val_accuracy: 0.8866 - val_loss: 0.4147
Epoch 24/50
469/469              3s 7ms/step -
accuracy: 0.8819 - loss: 0.4281 - val_accuracy: 0.8899 - val_loss: 0.4060
Epoch 25/50
469/469              3s 7ms/step -
accuracy: 0.8859 - loss: 0.4187 - val_accuracy: 0.8916 - val_loss: 0.3981
Epoch 26/50
469/469              4s 8ms/step -
accuracy: 0.8856 - loss: 0.4150 - val_accuracy: 0.8921 - val_loss: 0.3911
Epoch 27/50
469/469              3s 7ms/step -
accuracy: 0.8889 - loss: 0.4053 - val_accuracy: 0.8939 - val_loss: 0.3838
Epoch 28/50
469/469              3s 7ms/step -
accuracy: 0.8879 - loss: 0.3982 - val_accuracy: 0.8959 - val_loss: 0.3787
Epoch 29/50
```

```
469/469             5s 10ms/step -
accuracy: 0.8915 - loss: 0.3929 - val_accuracy: 0.8949 - val_loss: 0.3736
Epoch 30/50
469/469             3s 6ms/step -
accuracy: 0.8921 - loss: 0.3888 - val_accuracy: 0.8966 - val_loss: 0.3689
Epoch 31/50
469/469             3s 7ms/step -
accuracy: 0.8946 - loss: 0.3794 - val_accuracy: 0.8988 - val_loss: 0.3635
Epoch 32/50
469/469             4s 8ms/step -
accuracy: 0.8930 - loss: 0.3759 - val_accuracy: 0.8992 - val_loss: 0.3599
Epoch 33/50
469/469             3s 7ms/step -
accuracy: 0.8927 - loss: 0.3809 - val_accuracy: 0.9008 - val_loss: 0.3557
Epoch 34/50
469/469             3s 6ms/step -
accuracy: 0.8979 - loss: 0.3624 - val_accuracy: 0.9003 - val_loss: 0.3516
Epoch 35/50
469/469             3s 6ms/step -
accuracy: 0.8975 - loss: 0.3633 - val_accuracy: 0.9008 - val_loss: 0.3485
Epoch 36/50
469/469             3s 6ms/step -
accuracy: 0.8955 - loss: 0.3685 - val_accuracy: 0.9008 - val_loss: 0.3465
Epoch 37/50
469/469             3s 7ms/step -
accuracy: 0.9005 - loss: 0.3517 - val_accuracy: 0.9016 - val_loss: 0.3425
Epoch 38/50
469/469             4s 9ms/step -
accuracy: 0.9001 - loss: 0.3515 - val_accuracy: 0.9028 - val_loss: 0.3403
Epoch 39/50
469/469             5s 10ms/step -
accuracy: 0.9010 - loss: 0.3501 - val_accuracy: 0.9034 - val_loss: 0.3377
Epoch 40/50
469/469             4s 9ms/step -
accuracy: 0.8989 - loss: 0.3556 - val_accuracy: 0.9042 - val_loss: 0.3346
Epoch 41/50
469/469             4s 8ms/step -
accuracy: 0.9002 - loss: 0.3516 - val_accuracy: 0.9042 - val_loss: 0.3323
Epoch 42/50
469/469             3s 7ms/step -
accuracy: 0.9012 - loss: 0.3486 - val_accuracy: 0.9052 - val_loss: 0.3320
Epoch 43/50
469/469             3s 6ms/step -
accuracy: 0.9015 - loss: 0.3465 - val_accuracy: 0.9056 - val_loss: 0.3291
Epoch 44/50
469/469             3s 6ms/step -
accuracy: 0.9021 - loss: 0.3456 - val_accuracy: 0.9051 - val_loss: 0.3261
Epoch 45/50
```

```
469/469                3s 7ms/step -
accuracy: 0.9047 - loss: 0.3332 - val_accuracy: 0.9045 - val_loss: 0.3244
Epoch 46/50
469/469                4s 7ms/step -
accuracy: 0.9030 - loss: 0.3359 - val_accuracy: 0.9063 - val_loss: 0.3218
Epoch 47/50
469/469                3s 7ms/step -
accuracy: 0.9048 - loss: 0.3345 - val_accuracy: 0.9074 - val_loss: 0.3202
Epoch 48/50
469/469                3s 7ms/step -
accuracy: 0.9060 - loss: 0.3295 - val_accuracy: 0.9086 - val_loss: 0.3186
Epoch 49/50
469/469                3s 7ms/step -
accuracy: 0.9059 - loss: 0.3331 - val_accuracy: 0.9093 - val_loss: 0.3174
Epoch 50/50
469/469                4s 9ms/step -
accuracy: 0.9075 - loss: 0.3280 - val_accuracy: 0.9089 - val_loss: 0.3157
313/313                1s 5ms/step -
accuracy: 0.8966 - loss: 0.3592
Test loss: 0.3157140612602234
Test Accuracy:90.89%
```

[10]:
```
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label = 'val_loss')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```





•

0.3 Regularization Techniques

0.3.1 Using dropout(0.2)

[11]:
```
model = Sequential()
model.add(Dense(512, activation = 'relu',input_shape = (784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation = 'softmax'))
model.summary()
sgd1 = SGD(learning_rate=0.01)
model.compile(loss = 'categorical_crossentropy', optimizer = sgd1, metrics =␣
 ↪['accuracy'])
history = model.
 ↪fit(x_train,y_train,batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(x_test,y
score = model.evaluate(x_test,y_test_, verbose = 1)
print("Test loss:", score[0])
print(f"Test Accuracy:{score[1]*100:.2f}%")
```

```
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
dense_9 (Dense)	(None, 512)	401,920
dropout (Dropout)	(None, 512)	0
dense_10 (Dense)	(None, 512)	262,656

```
 dropout_1 (Dropout)              (None, 512)                        0

 dense_11 (Dense)                 (None, 10)                     5,130


 Total params: 669,706 (2.55 MB)

 Trainable params: 669,706 (2.55 MB)

 Non-trainable params: 0 (0.00 B)

Epoch 1/50
469/469             8s 14ms/step -
accuracy: 0.4983 - loss: 1.7143 - val_accuracy: 0.8674 - val_loss: 0.5604
Epoch 2/50
469/469             5s 10ms/step -
accuracy: 0.8367 - loss: 0.5906 - val_accuracy: 0.8942 - val_loss: 0.3843
Epoch 3/50
469/469             4s 9ms/step -
accuracy: 0.8694 - loss: 0.4529 - val_accuracy: 0.9083 - val_loss: 0.3272
Epoch 4/50
469/469             4s 9ms/step -
accuracy: 0.8891 - loss: 0.3852 - val_accuracy: 0.9164 - val_loss: 0.2954
Epoch 5/50
469/469             4s 8ms/step -
accuracy: 0.8969 - loss: 0.3535 - val_accuracy: 0.9227 - val_loss: 0.2733
Epoch 6/50
469/469             4s 8ms/step -
accuracy: 0.9035 - loss: 0.3264 - val_accuracy: 0.9254 - val_loss: 0.2565
Epoch 7/50
469/469             4s 8ms/step -
accuracy: 0.9127 - loss: 0.3057 - val_accuracy: 0.9301 - val_loss: 0.2421
Epoch 8/50
469/469             4s 8ms/step -
accuracy: 0.9179 - loss: 0.2835 - val_accuracy: 0.9342 - val_loss: 0.2294
Epoch 9/50
469/469             4s 8ms/step -
accuracy: 0.9208 - loss: 0.2781 - val_accuracy: 0.9366 - val_loss: 0.2189
Epoch 10/50
469/469             4s 8ms/step -
accuracy: 0.9258 - loss: 0.2575 - val_accuracy: 0.9385 - val_loss: 0.2095
Epoch 11/50
469/469             4s 8ms/step -
accuracy: 0.9290 - loss: 0.2486 - val_accuracy: 0.9410 - val_loss: 0.1999
Epoch 12/50
469/469             4s 7ms/step -
```

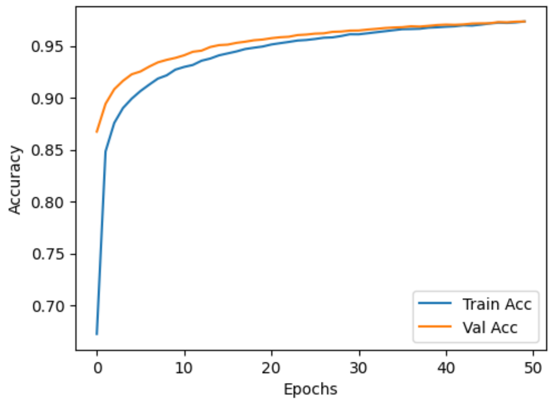
```
accuracy: 0.9304 - loss: 0.2378 - val_accuracy: 0.9444 - val_loss: 0.1907
Epoch 13/50
469/469            4s 8ms/step -
accuracy: 0.9351 - loss: 0.2267 - val_accuracy: 0.9455 - val_loss: 0.1842
Epoch 14/50
469/469            4s 8ms/step -
accuracy: 0.9370 - loss: 0.2153 - val_accuracy: 0.9490 - val_loss: 0.1772
Epoch 15/50
469/469            4s 8ms/step -
accuracy: 0.9389 - loss: 0.2102 - val_accuracy: 0.9508 - val_loss: 0.1708
Epoch 16/50
469/469            4s 9ms/step -
accuracy: 0.9417 - loss: 0.2029 - val_accuracy: 0.9513 - val_loss: 0.1658
Epoch 17/50
469/469            4s 8ms/step -
accuracy: 0.9449 - loss: 0.1921 - val_accuracy: 0.9530 - val_loss: 0.1594
Epoch 18/50
469/469            4s 9ms/step -
accuracy: 0.9464 - loss: 0.1857 - val_accuracy: 0.9541 - val_loss: 0.1545
Epoch 19/50
469/469            3s 7ms/step -
accuracy: 0.9484 - loss: 0.1794 - val_accuracy: 0.9555 - val_loss: 0.1495
Epoch 20/50
469/469            4s 8ms/step -
accuracy: 0.9484 - loss: 0.1775 - val_accuracy: 0.9562 - val_loss: 0.1447
Epoch 21/50
469/469            4s 8ms/step -
accuracy: 0.9509 - loss: 0.1704 - val_accuracy: 0.9575 - val_loss: 0.1408
Epoch 22/50
469/469            4s 8ms/step -
accuracy: 0.9530 - loss: 0.1635 - val_accuracy: 0.9584 - val_loss: 0.1371
Epoch 23/50
469/469            4s 8ms/step -
accuracy: 0.9534 - loss: 0.1632 - val_accuracy: 0.9589 - val_loss: 0.1351
Epoch 24/50
469/469            4s 8ms/step -
accuracy: 0.9546 - loss: 0.1564 - val_accuracy: 0.9605 - val_loss: 0.1312
Epoch 25/50
469/469            4s 8ms/step -
accuracy: 0.9562 - loss: 0.1542 - val_accuracy: 0.9610 - val_loss: 0.1277
Epoch 26/50
469/469            4s 8ms/step -
accuracy: 0.9569 - loss: 0.1489 - val_accuracy: 0.9619 - val_loss: 0.1256
Epoch 27/50
469/469            5s 10ms/step -
accuracy: 0.9565 - loss: 0.1482 - val_accuracy: 0.9622 - val_loss: 0.1221
Epoch 28/50
469/469            4s 8ms/step -
```

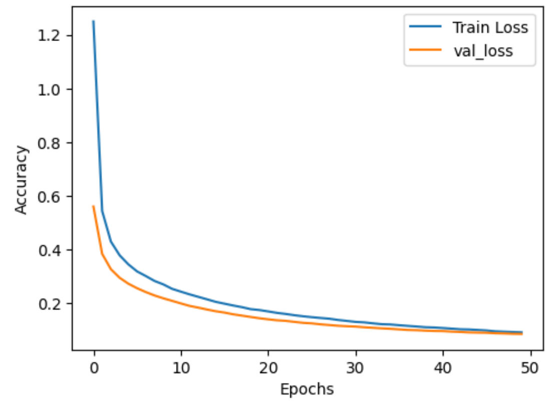
```
accuracy: 0.9577 - loss: 0.1424 - val_accuracy: 0.9636 - val_loss: 0.1195
Epoch 29/50
469/469            4s 8ms/step -
accuracy: 0.9592 - loss: 0.1391 - val_accuracy: 0.9639 - val_loss: 0.1168
Epoch 30/50
469/469            4s 8ms/step -
accuracy: 0.9626 - loss: 0.1319 - val_accuracy: 0.9648 - val_loss: 0.1150
Epoch 31/50
469/469            4s 8ms/step -
accuracy: 0.9612 - loss: 0.1313 - val_accuracy: 0.9649 - val_loss: 0.1137
Epoch 32/50
469/469            4s 8ms/step -
accuracy: 0.9601 - loss: 0.1348 - val_accuracy: 0.9658 - val_loss: 0.1111
Epoch 33/50
469/469            4s 8ms/step -
accuracy: 0.9637 - loss: 0.1251 - val_accuracy: 0.9665 - val_loss: 0.1092
Epoch 34/50
469/469            4s 7ms/step -
accuracy: 0.9633 - loss: 0.1246 - val_accuracy: 0.9673 - val_loss: 0.1071
Epoch 35/50
469/469            4s 7ms/step -
accuracy: 0.9646 - loss: 0.1237 - val_accuracy: 0.9678 - val_loss: 0.1054
Epoch 36/50
469/469            4s 8ms/step -
accuracy: 0.9652 - loss: 0.1221 - val_accuracy: 0.9681 - val_loss: 0.1033
Epoch 37/50
469/469            4s 9ms/step -
accuracy: 0.9661 - loss: 0.1181 - val_accuracy: 0.9690 - val_loss: 0.1011
Epoch 38/50
469/469            4s 8ms/step -
accuracy: 0.9662 - loss: 0.1148 - val_accuracy: 0.9687 - val_loss: 0.1002
Epoch 39/50
469/469            4s 8ms/step -
accuracy: 0.9673 - loss: 0.1109 - val_accuracy: 0.9694 - val_loss: 0.0985
Epoch 40/50
469/469            4s 7ms/step -
accuracy: 0.9681 - loss: 0.1125 - val_accuracy: 0.9702 - val_loss: 0.0974
Epoch 41/50
469/469            4s 8ms/step -
accuracy: 0.9679 - loss: 0.1084 - val_accuracy: 0.9706 - val_loss: 0.0969
Epoch 42/50
469/469            4s 8ms/step -
accuracy: 0.9703 - loss: 0.1026 - val_accuracy: 0.9704 - val_loss: 0.0950
Epoch 43/50
469/469            4s 8ms/step -
accuracy: 0.9698 - loss: 0.1040 - val_accuracy: 0.9708 - val_loss: 0.0937
Epoch 44/50
469/469            4s 8ms/step -
```

```
accuracy: 0.9700 - loss: 0.1033 - val_accuracy: 0.9716 - val_loss: 0.0920
Epoch 45/50
469/469            4s 8ms/step -
accuracy: 0.9702 - loss: 0.1034 - val_accuracy: 0.9718 - val_loss: 0.0914
Epoch 46/50
469/469            3s 7ms/step -
accuracy: 0.9720 - loss: 0.0968 - val_accuracy: 0.9719 - val_loss: 0.0907
Epoch 47/50
469/469            4s 8ms/step -
accuracy: 0.9726 - loss: 0.0963 - val_accuracy: 0.9730 - val_loss: 0.0890
Epoch 48/50
469/469            4s 8ms/step -
accuracy: 0.9720 - loss: 0.0955 - val_accuracy: 0.9728 - val_loss: 0.0881
Epoch 49/50
469/469            4s 8ms/step -
accuracy: 0.9723 - loss: 0.0944 - val_accuracy: 0.9734 - val_loss: 0.0869
Epoch 50/50
469/469            4s 8ms/step -
accuracy: 0.9739 - loss: 0.0916 - val_accuracy: 0.9736 - val_loss: 0.0864
313/313            1s 3ms/step -
accuracy: 0.9689 - loss: 0.1017
Test loss: 0.08635895699262619
Test Accuracy:97.36%
```

[12]:

```
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label = 'val_loss')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```





0.4 Early stopping

[14]:
```python
from sklearn.model_selection import train_test_split
(x_train,y_train),(x_test,y_test) = mnist.load_data()
x_subtrain,x_valid,y_subtrain,y_valid = train_test_split(x_train,y_train,test_size = 0.10, random_state = 1)
x_train = x_train/255
x_test = x_test/255
x_subtrain = x_subtrain/255
x_valid=x_valid/255
```

0.4.1 making the ANN

[15]:
```python
model = Sequential()
model.add(Flatten(input_shape = (28,28)))
model.add(Dense(512, activation = 'relu'))
model.add(Dense(512, activation = 'relu'))
model.add(Dense(10, activation = 'softmax'))
model.summary()
sgd1 = SGD(learning_rate=0.01)
```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense_12 (Dense)	(None, 512)	401,920
dense_13 (Dense)	(None, 512)	262,656

```
 dense_14 (Dense)                  (None, 10)                     5,130
```

Total params: 669,706 (2.55 MB)

Trainable params: 669,706 (2.55 MB)

Non-trainable params: 0 (0.00 B)

[16]:
```
from keras.callbacks import EarlyStopping
model.compile(loss= 'sparse_categorical_crossentropy',optimizer = sgd1,metrics␣
 ↪= ['accuracy'])
estop = EarlyStopping(monitor = 'val_loss', min_delta = 1e-3, mode = 'min',␣
 ↪patience = 4, verbose = 1, restore_best_weights= True)
history = model.fit(x_subtrain,y_subtrain, batch_size=batch_size, epochs = 100,␣
 ↪verbose = 1, validation_data=(x_valid,y_valid))
```

```
Epoch 1/100
422/422             4s 8ms/step -
accuracy: 0.5949 - loss: 1.6476 - val_accuracy: 0.8600 - val_loss: 0.5826
Epoch 2/100
422/422             3s 7ms/step -
accuracy: 0.8701 - loss: 0.5185 - val_accuracy: 0.8885 - val_loss: 0.4104
Epoch 3/100
422/422             3s 7ms/step -
accuracy: 0.8935 - loss: 0.3859 - val_accuracy: 0.9037 - val_loss: 0.3525
Epoch 4/100
422/422             3s 6ms/step -
accuracy: 0.9050 - loss: 0.3394 - val_accuracy: 0.9103 - val_loss: 0.3219
Epoch 5/100
422/422             3s 7ms/step -
accuracy: 0.9129 - loss: 0.3069 - val_accuracy: 0.9160 - val_loss: 0.3012
Epoch 6/100
422/422             3s 6ms/step -
accuracy: 0.9180 - loss: 0.2882 - val_accuracy: 0.9188 - val_loss: 0.2832
Epoch 7/100
422/422             3s 6ms/step -
accuracy: 0.9247 - loss: 0.2650 - val_accuracy: 0.9240 - val_loss: 0.2690
Epoch 8/100
422/422             3s 6ms/step -
accuracy: 0.9286 - loss: 0.2531 - val_accuracy: 0.9263 - val_loss: 0.2579
Epoch 9/100
422/422             3s 7ms/step -
accuracy: 0.9307 - loss: 0.2442 - val_accuracy: 0.9292 - val_loss: 0.2479
Epoch 10/100
```

```
422/422            3s 6ms/step -
accuracy: 0.9343 - loss: 0.2264 - val_accuracy: 0.9320 - val_loss: 0.2387
Epoch 11/100
422/422            3s 7ms/step -
accuracy: 0.9356 - loss: 0.2237 - val_accuracy: 0.9342 - val_loss: 0.2303
Epoch 12/100
422/422            3s 7ms/step -
accuracy: 0.9414 - loss: 0.2073 - val_accuracy: 0.9352 - val_loss: 0.2252
Epoch 13/100
422/422            3s 7ms/step -
accuracy: 0.9425 - loss: 0.2027 - val_accuracy: 0.9365 - val_loss: 0.2181
Epoch 14/100
422/422            3s 6ms/step -
accuracy: 0.9439 - loss: 0.1951 - val_accuracy: 0.9407 - val_loss: 0.2099
Epoch 15/100
422/422            3s 6ms/step -
accuracy: 0.9447 - loss: 0.1906 - val_accuracy: 0.9418 - val_loss: 0.2042
Epoch 16/100
422/422            3s 6ms/step -
accuracy: 0.9497 - loss: 0.1794 - val_accuracy: 0.9432 - val_loss: 0.2003
Epoch 17/100
422/422            3s 6ms/step -
accuracy: 0.9509 - loss: 0.1740 - val_accuracy: 0.9430 - val_loss: 0.1951
Epoch 18/100
422/422            3s 6ms/step -
accuracy: 0.9502 - loss: 0.1740 - val_accuracy: 0.9448 - val_loss: 0.1894
Epoch 19/100
422/422            3s 6ms/step -
accuracy: 0.9535 - loss: 0.1615 - val_accuracy: 0.9470 - val_loss: 0.1845
Epoch 20/100
422/422            3s 6ms/step -
accuracy: 0.9550 - loss: 0.1623 - val_accuracy: 0.9467 - val_loss: 0.1825
Epoch 21/100
422/422            3s 7ms/step -
accuracy: 0.9559 - loss: 0.1563 - val_accuracy: 0.9488 - val_loss: 0.1756
Epoch 22/100
422/422            3s 7ms/step -
accuracy: 0.9576 - loss: 0.1476 - val_accuracy: 0.9500 - val_loss: 0.1722
Epoch 23/100
422/422            3s 7ms/step -
accuracy: 0.9594 - loss: 0.1424 - val_accuracy: 0.9500 - val_loss: 0.1687
Epoch 24/100
422/422            3s 6ms/step -
accuracy: 0.9604 - loss: 0.1405 - val_accuracy: 0.9517 - val_loss: 0.1661
Epoch 25/100
422/422            3s 6ms/step -
accuracy: 0.9612 - loss: 0.1374 - val_accuracy: 0.9513 - val_loss: 0.1638
Epoch 26/100
```

```
422/422             3s 6ms/step -
accuracy: 0.9627 - loss: 0.1315 - val_accuracy: 0.9520 - val_loss: 0.1595
Epoch 27/100
422/422             3s 6ms/step -
accuracy: 0.9639 - loss: 0.1278 - val_accuracy: 0.9532 - val_loss: 0.1580
Epoch 28/100
422/422             3s 6ms/step -
accuracy: 0.9633 - loss: 0.1277 - val_accuracy: 0.9548 - val_loss: 0.1562
Epoch 29/100
422/422             3s 6ms/step -
accuracy: 0.9652 - loss: 0.1215 - val_accuracy: 0.9567 - val_loss: 0.1515
Epoch 30/100
422/422             3s 6ms/step -
accuracy: 0.9666 - loss: 0.1195 - val_accuracy: 0.9565 - val_loss: 0.1488
Epoch 31/100
422/422             3s 7ms/step -
accuracy: 0.9669 - loss: 0.1175 - val_accuracy: 0.9577 - val_loss: 0.1477
Epoch 32/100
422/422             3s 7ms/step -
accuracy: 0.9680 - loss: 0.1117 - val_accuracy: 0.9577 - val_loss: 0.1450
Epoch 33/100
422/422             3s 6ms/step -
accuracy: 0.9684 - loss: 0.1121 - val_accuracy: 0.9582 - val_loss: 0.1429
Epoch 34/100
422/422             3s 6ms/step -
accuracy: 0.9695 - loss: 0.1081 - val_accuracy: 0.9590 - val_loss: 0.1402
Epoch 35/100
422/422             3s 7ms/step -
accuracy: 0.9698 - loss: 0.1077 - val_accuracy: 0.9597 - val_loss: 0.1391
Epoch 36/100
422/422             3s 7ms/step -
accuracy: 0.9696 - loss: 0.1061 - val_accuracy: 0.9615 - val_loss: 0.1364
Epoch 37/100
422/422             3s 7ms/step -
accuracy: 0.9712 - loss: 0.1042 - val_accuracy: 0.9607 - val_loss: 0.1367
Epoch 38/100
422/422             3s 6ms/step -
accuracy: 0.9718 - loss: 0.0989 - val_accuracy: 0.9625 - val_loss: 0.1335
Epoch 39/100
422/422             3s 6ms/step -
accuracy: 0.9735 - loss: 0.0974 - val_accuracy: 0.9628 - val_loss: 0.1310
Epoch 40/100
422/422             3s 7ms/step -
accuracy: 0.9733 - loss: 0.0936 - val_accuracy: 0.9630 - val_loss: 0.1294
Epoch 41/100
422/422             3s 7ms/step -
accuracy: 0.9742 - loss: 0.0947 - val_accuracy: 0.9622 - val_loss: 0.1285
Epoch 42/100
```

```
422/422            3s 7ms/step -
accuracy: 0.9758 - loss: 0.0882 - val_accuracy: 0.9623 - val_loss: 0.1285
Epoch 43/100
422/422            3s 6ms/step -
accuracy: 0.9755 - loss: 0.0873 - val_accuracy: 0.9628 - val_loss: 0.1256
Epoch 44/100
422/422            3s 6ms/step -
accuracy: 0.9759 - loss: 0.0872 - val_accuracy: 0.9643 - val_loss: 0.1240
Epoch 45/100
422/422            3s 8ms/step -
accuracy: 0.9764 - loss: 0.0849 - val_accuracy: 0.9650 - val_loss: 0.1223
Epoch 46/100
422/422            3s 7ms/step -
accuracy: 0.9770 - loss: 0.0832 - val_accuracy: 0.9645 - val_loss: 0.1214
Epoch 47/100
422/422            3s 6ms/step -
accuracy: 0.9771 - loss: 0.0827 - val_accuracy: 0.9653 - val_loss: 0.1198
Epoch 48/100
422/422            3s 7ms/step -
accuracy: 0.9782 - loss: 0.0794 - val_accuracy: 0.9660 - val_loss: 0.1192
Epoch 49/100
422/422            3s 7ms/step -
accuracy: 0.9795 - loss: 0.0747 - val_accuracy: 0.9672 - val_loss: 0.1174
Epoch 50/100
422/422            3s 7ms/step -
accuracy: 0.9788 - loss: 0.0787 - val_accuracy: 0.9663 - val_loss: 0.1168
Epoch 51/100
422/422            3s 7ms/step -
accuracy: 0.9802 - loss: 0.0742 - val_accuracy: 0.9668 - val_loss: 0.1151
Epoch 52/100
422/422            3s 7ms/step -
accuracy: 0.9799 - loss: 0.0743 - val_accuracy: 0.9677 - val_loss: 0.1139
Epoch 53/100
422/422            3s 7ms/step -
accuracy: 0.9800 - loss: 0.0716 - val_accuracy: 0.9670 - val_loss: 0.1134
Epoch 54/100
422/422            3s 7ms/step -
accuracy: 0.9814 - loss: 0.0704 - val_accuracy: 0.9673 - val_loss: 0.1128
Epoch 55/100
422/422            3s 7ms/step -
accuracy: 0.9820 - loss: 0.0676 - val_accuracy: 0.9683 - val_loss: 0.1110
Epoch 56/100
422/422            3s 7ms/step -
accuracy: 0.9808 - loss: 0.0704 - val_accuracy: 0.9687 - val_loss: 0.1108
Epoch 57/100
422/422            3s 6ms/step -
accuracy: 0.9821 - loss: 0.0655 - val_accuracy: 0.9680 - val_loss: 0.1100
Epoch 58/100
```

```
422/422            3s 7ms/step -
accuracy: 0.9832 - loss: 0.0634 - val_accuracy: 0.9695 - val_loss: 0.1086
Epoch 59/100
422/422            3s 7ms/step -
accuracy: 0.9826 - loss: 0.0642 - val_accuracy: 0.9683 - val_loss: 0.1074
Epoch 60/100
422/422            3s 7ms/step -
accuracy: 0.9844 - loss: 0.0609 - val_accuracy: 0.9692 - val_loss: 0.1070
Epoch 61/100
422/422            3s 7ms/step -
accuracy: 0.9846 - loss: 0.0594 - val_accuracy: 0.9697 - val_loss: 0.1060
Epoch 62/100
422/422            3s 7ms/step -
accuracy: 0.9844 - loss: 0.0618 - val_accuracy: 0.9695 - val_loss: 0.1049
Epoch 63/100
422/422            3s 7ms/step -
accuracy: 0.9851 - loss: 0.0592 - val_accuracy: 0.9700 - val_loss: 0.1050
Epoch 64/100
422/422            3s 8ms/step -
accuracy: 0.9853 - loss: 0.0576 - val_accuracy: 0.9698 - val_loss: 0.1045
Epoch 65/100
422/422            5s 8ms/step -
accuracy: 0.9847 - loss: 0.0588 - val_accuracy: 0.9712 - val_loss: 0.1029
Epoch 66/100
422/422            3s 6ms/step -
accuracy: 0.9863 - loss: 0.0546 - val_accuracy: 0.9703 - val_loss: 0.1029
Epoch 67/100
422/422            3s 7ms/step -
accuracy: 0.9866 - loss: 0.0537 - val_accuracy: 0.9705 - val_loss: 0.1019
Epoch 68/100
422/422            3s 7ms/step -
accuracy: 0.9861 - loss: 0.0528 - val_accuracy: 0.9712 - val_loss: 0.1018
Epoch 69/100
422/422            3s 7ms/step -
accuracy: 0.9863 - loss: 0.0525 - val_accuracy: 0.9700 - val_loss: 0.1011
Epoch 70/100
422/422            3s 7ms/step -
accuracy: 0.9869 - loss: 0.0524 - val_accuracy: 0.9708 - val_loss: 0.1009
Epoch 71/100
422/422            3s 7ms/step -
accuracy: 0.9861 - loss: 0.0531 - val_accuracy: 0.9715 - val_loss: 0.1000
Epoch 72/100
422/422            3s 7ms/step -
accuracy: 0.9871 - loss: 0.0513 - val_accuracy: 0.9722 - val_loss: 0.0985
Epoch 73/100
422/422            3s 7ms/step -
accuracy: 0.9874 - loss: 0.0490 - val_accuracy: 0.9725 - val_loss: 0.0987
Epoch 74/100
```

```
422/422 3s 7ms/step -
accuracy: 0.9882 - loss: 0.0483 - val_accuracy: 0.9715 - val_loss: 0.0998
Epoch 75/100
422/422 3s 7ms/step -
accuracy: 0.9867 - loss: 0.0514 - val_accuracy: 0.9723 - val_loss: 0.0991
Epoch 76/100
422/422 3s 7ms/step -
accuracy: 0.9879 - loss: 0.0476 - val_accuracy: 0.9707 - val_loss: 0.0981
Epoch 77/100
422/422 3s 7ms/step -
accuracy: 0.9888 - loss: 0.0449 - val_accuracy: 0.9720 - val_loss: 0.0979
Epoch 78/100
422/422 3s 7ms/step -
accuracy: 0.9896 - loss: 0.0443 - val_accuracy: 0.9728 - val_loss: 0.0965
Epoch 79/100
422/422 3s 7ms/step -
accuracy: 0.9895 - loss: 0.0434 - val_accuracy: 0.9725 - val_loss: 0.0954
Epoch 80/100
422/422 3s 7ms/step -
accuracy: 0.9896 - loss: 0.0426 - val_accuracy: 0.9722 - val_loss: 0.0957
Epoch 81/100
422/422 3s 7ms/step -
accuracy: 0.9887 - loss: 0.0450 - val_accuracy: 0.9727 - val_loss: 0.0960
Epoch 82/100
422/422 3s 7ms/step -
accuracy: 0.9895 - loss: 0.0442 - val_accuracy: 0.9728 - val_loss: 0.0947
Epoch 83/100
422/422 3s 7ms/step -
accuracy: 0.9900 - loss: 0.0411 - val_accuracy: 0.9735 - val_loss: 0.0941
Epoch 84/100
422/422 3s 6ms/step -
accuracy: 0.9903 - loss: 0.0421 - val_accuracy: 0.9730 - val_loss: 0.0944
Epoch 85/100
422/422 3s 7ms/step -
accuracy: 0.9893 - loss: 0.0421 - val_accuracy: 0.9727 - val_loss: 0.0933
Epoch 86/100
422/422 4s 8ms/step -
accuracy: 0.9906 - loss: 0.0402 - val_accuracy: 0.9725 - val_loss: 0.0942
Epoch 87/100
422/422 3s 7ms/step -
accuracy: 0.9906 - loss: 0.0391 - val_accuracy: 0.9728 - val_loss: 0.0937
Epoch 88/100
422/422 3s 7ms/step -
accuracy: 0.9909 - loss: 0.0397 - val_accuracy: 0.9725 - val_loss: 0.0935
Epoch 89/100
422/422 4s 9ms/step -
accuracy: 0.9912 - loss: 0.0378 - val_accuracy: 0.9733 - val_loss: 0.0932
Epoch 90/100
```

```
422/422                3s 8ms/step -
accuracy: 0.9915 - loss: 0.0368 - val_accuracy: 0.9727 - val_loss: 0.0923
Epoch 91/100
422/422                3s 8ms/step -
accuracy: 0.9909 - loss: 0.0367 - val_accuracy: 0.9730 - val_loss: 0.0925
Epoch 92/100
422/422                4s 9ms/step -
accuracy: 0.9918 - loss: 0.0351 - val_accuracy: 0.9742 - val_loss: 0.0919
Epoch 93/100
422/422                3s 8ms/step -
accuracy: 0.9915 - loss: 0.0359 - val_accuracy: 0.9733 - val_loss: 0.0913
Epoch 94/100
422/422                4s 8ms/step -
accuracy: 0.9917 - loss: 0.0354 - val_accuracy: 0.9738 - val_loss: 0.0914
Epoch 95/100
422/422                4s 10ms/step -
accuracy: 0.9921 - loss: 0.0340 - val_accuracy: 0.9735 - val_loss: 0.0910
Epoch 96/100
422/422                4s 9ms/step -
accuracy: 0.9931 - loss: 0.0326 - val_accuracy: 0.9737 - val_loss: 0.0909
Epoch 97/100
422/422                3s 7ms/step -
accuracy: 0.9933 - loss: 0.0315 - val_accuracy: 0.9733 - val_loss: 0.0904
Epoch 98/100
422/422                3s 7ms/step -
accuracy: 0.9928 - loss: 0.0329 - val_accuracy: 0.9738 - val_loss: 0.0915
Epoch 99/100
422/422                3s 7ms/step -
accuracy: 0.9925 - loss: 0.0334 - val_accuracy: 0.9738 - val_loss: 0.0900
Epoch 100/100
422/422                4s 8ms/step -
accuracy: 0.9934 - loss: 0.0309 - val_accuracy: 0.9743 - val_loss: 0.0898
```

[17]:

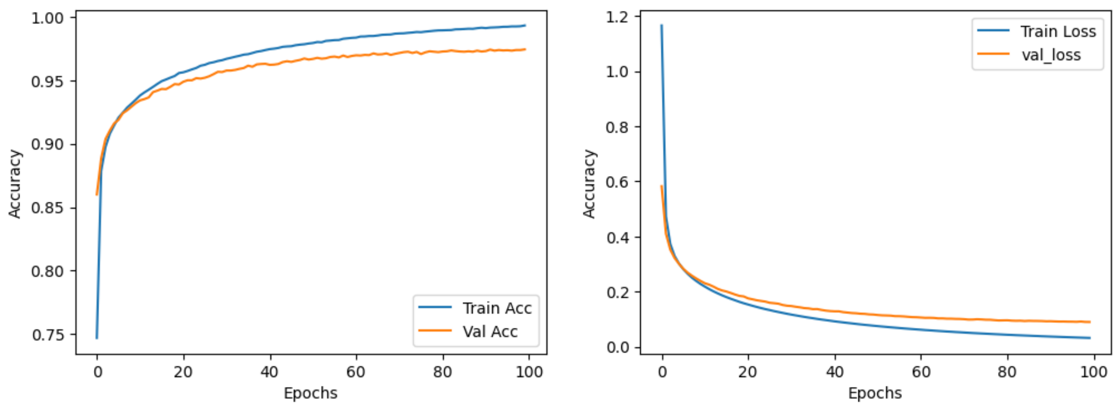
```
score = model.evaluate(x_test,y_test, verbose = 1)
print("Test loss:", score[0])
print(f"Test Accuracy:{score[1]*100:.2f}%")
```

```
313/313                1s 3ms/step -
accuracy: 0.9736 - loss: 0.0866
Test loss: 0.07452523708343506
Test Accuracy:97.74%
```

[18]:

```
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
```

```
plt.legend()
plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label = 'val_loss')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



0.5 Challenging Question: Try for a scratch code for this case where you can create a custom neural network without using any inbuilt classes like sequential etc. Where you need to define a class neural network which has methods like forwardpass, backwardpass, and train. Figure out how we can do this. This model has inputs as [0, 0, 1], [0, 1, 1], [1, 0, 1], [1, 1, 1] and the expected output as [0], [1], [1], [0] in each case. So there are three features in our dataset as you see above. The activation function is to be taken as sigmoid. The architecture is like we have only one hidden layer and an output layer with one neuron. Take the error function as (1/2)(y − y^)^2

[19]:
```
import numpy as np

# sigmoid activation function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# derivative of sigmoid function
def sigmoid_derivative(x):
    return x*(1-x)

# mean squared error loss
def mse_loss(y_true,y_pred):
    return 0.5*np.mean((y_true-y_pred)**2)
```

```python
# Input dataset (XOR gate inputs with bias term)
x = np.array([[0,0,1],
[0,1,1],
[1,0,1],
[1,1,1]])

# output labels
y = np.array([[0],
[1],
[1],
[0]])

# seed for reproducibility
np.random.seed(1)

# Initialize weights randomly with mean 0
input_size = 3 # 3 input features
hidden_size = 2 # 2 hidden layers
output_size = 1 # 1 output neuron
# Weights
w1 = 2 * np.random.random((input_size, hidden_size))-1
w2 = 2 * np.random.random((hidden_size, output_size))-1
# Biases
b1 = np.zeros((1, hidden_size))
b2 = np.zeros((1, output_size))
# Learning rate
lr = 0.1
# Training loop

for epoch in range(10000):
    ##------------- Forward pass -------------------
    a1 = np.dot(x,w1) + b1
    h1 = sigmoid(a1) # activation of hidden layer
    a2 = np.dot(h1,w2) + b2
    output = sigmoid(a2) # final prediction
    # loss calculation
    loss = mse_loss(y,output)
    ##-------------- Back propagation ---------------
    # output layer error
    output_error = output - y
    output_delta = output_error * sigmoid_derivative(output)
    ## hidden layer error
    hidden_error = np.dot(output_delta, w2.T)
    hidden_delta = hidden_error * sigmoid_derivative(h1)
    ##------------Updating weights and biases -----------
    w2 -= lr * np.dot(h1.T,output_delta)
```

```python
    b2 -= lr * np.sum(output_delta, axis = 0, keepdims = True)
    w1 -= lr * np.dot(x.T, hidden_delta)
    b1 -= lr * np.sum(hidden_delta, axis = 0, keepdims = True)
    # Print loss every 1000 epochs
    if epoch % 1000 == 0:
        print(f"Epoch {epoch}, Loss: {loss:.4f}")
# --------- Final Output ---------
print("\nFinal predictions after training:")
print(output.round(3))
```

```
Epoch 0, Loss: 0.1267
Epoch 1000, Loss: 0.1215
Epoch 2000, Loss: 0.1029
Epoch 3000, Loss: 0.0905
Epoch 4000, Loss: 0.0828
Epoch 5000, Loss: 0.0433
Epoch 6000, Loss: 0.0105
Epoch 7000, Loss: 0.0049
Epoch 8000, Loss: 0.0031
Epoch 9000, Loss: 0.0022

Final predictions after training:
[[0.049]
 [0.945]
 [0.945]
 [0.071]]
```

0.6 Question2.

0.6.1 cifar10 dataset is also an inbuilt dataset which contains 10 classes of images, mainly, 0-airplane, 1-automobile, 2-bird, 3-cat, 4-deer, 5-dog, 6-frog, 7-horse, 8-ship, 9-truck.Load the inbuilt dataset cifar10 as you did in last lab by replacing mnist.load datae() as cifar10.load data(). First, try to import it from keras.datasets as you did for mnist. Now, identify the size of the images you have first of all. You can now see 32 * 32 * 3 images that is 32* 32 pixel images with 3 channels that give the RGB values since we have a color image. Try to print the shape of each image and see. you will see it's stored like 32 * 32 *3 arrays. Now, try to visualize certain images using appropriate functions. Check the size of x train and x test and reshape them into one-dimensional arrays as done in the case of mnist dataset. Do necessary pre-processing and split the data into training, validation, and testing sets. Create a new model using a sequential class with appropriate hidden layers and output layer neurons. Choose appropriate activation functions like sigmoid and relu, etc. And also an appropriate one in the output layer. Choose the error function appropriately. Include early stopping technique in your model and run the model for 500 epochs. Try to come up with a better model with decent accuracy.The choice we have taken in the model here may not be the appropriate one. But you can see the accuracy you are able to come up with without having overfitting happen there.

0.6.2 Importing the necessary libraries

[20]:
```
import keras
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense,Dropout,Flatten
from keras.optimizers import Adam,SGD
import matplotlib.pyplot as plt
from keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from keras.regularizers import l2
import warnings
warnings.filterwarnings('ignore')
```

0.6.3 Loading the dataset and validation split

[21]:
```
(x_train,y_train), (x_test,y_test) = cifar10.load_data()
x_subtrain,x_valid,y_subtrain,y_valid =␣
 ↪train_test_split(x_train,y_train,test_size = 0.10, random_state = 1)
```

[22]:
```
x_subtrain.shape
```

[22]:
```
(45000, 32, 32, 3)
```

[23]:
```
x_valid.shape
```

[23]: (5000, 32, 32, 3)

[24]:
```
x_train = x_train/255
x_test = x_test/255
x_subtrain = x_subtrain/255
x_valid=x_valid/255
```

[25]:
```
# Flattening images
x_subtrain_flat = x_subtrain.reshape(x_subtrain.shape[0], -1)
x_valid_flat = x_valid.reshape(x_valid.shape[0], -1)
x_test_flat = x_test.reshape(x_test.shape[0], -1)
```

0.6.4 Some random images from the dataset along with their labels

[26]:
```
import numpy as np

# Mapping of label numbers to class names
label_names = {
    0: "airplane",
    1: "automobile",
    2: "bird",
    3: "cat",
    4: "deer",
    5: "dog",
    6: "frog",
    7: "horse",
    8: "ship",
    9: "truck"
}

# Picking 5 random indexes
random_indices = np.random.choice(len(x_train), size=5, replace=False)

# Plotting the images with labels
plt.figure(figsize=(2, 10))
for i, idx in enumerate(random_indices):
    plt.subplot(5, 1, i+1)
    plt.imshow(x_train[idx])
    plt.title(label_names[int(y_train[idx])])
    plt.axis('off')

plt.tight_layout()
plt.show()
```


cat
horse
bird
airplane
frog

0.6.5 One hot encoding the target labels

[27]:
```
y_subtrain_cat = to_categorical(y_subtrain, 10)
y_valid_cat = to_categorical(y_valid, 10)
y_test_cat = to_categorical(y_test, 10)
```

0.6.6 Building the model architecture

- without regularization or dropout layer

[28]:
```
model = Sequential()
model.add(Dense(512, activation = 'relu',input_shape = (3072,)))
model.add(Dense(256, activation = 'relu'))
model.add(Dense(128, activation = 'relu'))
model.add(Dense(10, activation = 'softmax'))
model.summary()
```

```
Model: "sequential_5"


 Layer (type)                    Output Shape                 Param #

 dense_15 (Dense)                (None, 512)                1,573,376

 dense_16 (Dense)                (None, 256)                  131,328

 dense_17 (Dense)                (None, 128)                   32,896

 dense_18 (Dense)                (None, 10)                     1,290


 Total params: 1,738,890 (6.63 MB)

 Trainable params: 1,738,890 (6.63 MB)

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

0.6.7 Compiling and running the model

[29]:
```
from keras.callbacks import EarlyStopping
```

```
model.compile(loss= 'categorical_crossentropy',optimizer = Adam(learning_rate=0.
    ↪001),metrics = ['accuracy'])

estop = EarlyStopping(monitor = 'val_loss', min_delta = 1e-4, mode = 'min',␣
    ↪patience = 5, verbose = 1, restore_best_weights=True)
history = model.fit(x_subtrain_flat,y_subtrain_cat, batch_size=128, epochs =␣
    ↪500, verbose = 1, validation_data=(x_valid_flat,y_valid_cat),␣
    ↪callbacks=[estop])
```

```
Epoch 1/500
352/352              9s 22ms/step -
accuracy: 0.2554 - loss: 2.0817 - val_accuracy: 0.3670 - val_loss: 1.7583
Epoch 2/500
352/352              8s 23ms/step -
accuracy: 0.3826 - loss: 1.7192 - val_accuracy: 0.4096 - val_loss: 1.6545
Epoch 3/500
352/352              7s 20ms/step -
accuracy: 0.4096 - loss: 1.6392 - val_accuracy: 0.4168 - val_loss: 1.6265
Epoch 4/500
352/352              7s 20ms/step -
accuracy: 0.4449 - loss: 1.5593 - val_accuracy: 0.4224 - val_loss: 1.6146
Epoch 5/500
352/352              7s 21ms/step -
accuracy: 0.4554 - loss: 1.5212 - val_accuracy: 0.4634 - val_loss: 1.5215
Epoch 6/500
352/352              7s 20ms/step -
accuracy: 0.4682 - loss: 1.4804 - val_accuracy: 0.4612 - val_loss: 1.5085
Epoch 7/500
352/352              7s 20ms/step -
accuracy: 0.4845 - loss: 1.4437 - val_accuracy: 0.4658 - val_loss: 1.4851
Epoch 8/500
352/352              7s 21ms/step -
accuracy: 0.4944 - loss: 1.4155 - val_accuracy: 0.4784 - val_loss: 1.4744
Epoch 9/500
352/352              9s 25ms/step -
accuracy: 0.5056 - loss: 1.3740 - val_accuracy: 0.4386 - val_loss: 1.5614
Epoch 10/500
352/352              8s 22ms/step -
accuracy: 0.5130 - loss: 1.3540 - val_accuracy: 0.4904 - val_loss: 1.4252
Epoch 11/500
352/352              7s 20ms/step -
accuracy: 0.5279 - loss: 1.3258 - val_accuracy: 0.4926 - val_loss: 1.4139
Epoch 12/500
352/352              7s 21ms/step -
accuracy: 0.5347 - loss: 1.3026 - val_accuracy: 0.4996 - val_loss: 1.4141
Epoch 13/500
352/352              7s 21ms/step -
```

```
accuracy: 0.5466 - loss: 1.2611 - val_accuracy: 0.5052 - val_loss: 1.4028
Epoch 14/500
352/352              9s 25ms/step -
accuracy: 0.5610 - loss: 1.2322 - val_accuracy: 0.5026 - val_loss: 1.3980
Epoch 15/500
352/352              8s 23ms/step -
accuracy: 0.5694 - loss: 1.2097 - val_accuracy: 0.5058 - val_loss: 1.4046
Epoch 16/500
352/352              7s 21ms/step -
accuracy: 0.5741 - loss: 1.1996 - val_accuracy: 0.5158 - val_loss: 1.3886
Epoch 17/500
352/352              8s 23ms/step -
accuracy: 0.5836 - loss: 1.1684 - val_accuracy: 0.5126 - val_loss: 1.3907
Epoch 18/500
352/352              8s 23ms/step -
accuracy: 0.5900 - loss: 1.1437 - val_accuracy: 0.5034 - val_loss: 1.4234
Epoch 19/500
352/352              7s 21ms/step -
accuracy: 0.5990 - loss: 1.1282 - val_accuracy: 0.5032 - val_loss: 1.4273
Epoch 20/500
352/352              7s 20ms/step -
accuracy: 0.6108 - loss: 1.0971 - val_accuracy: 0.5144 - val_loss: 1.4316
Epoch 21/500
352/352              7s 19ms/step -
accuracy: 0.6178 - loss: 1.0706 - val_accuracy: 0.5052 - val_loss: 1.4354
Epoch 21: early stopping
Restoring model weights from the end of the best epoch: 16.
```

[30]:

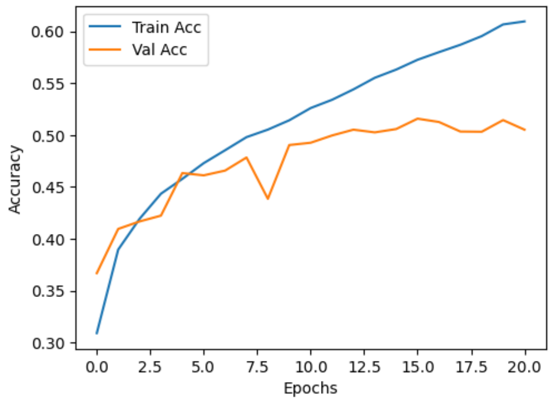
```
score = model.evaluate(x_test_flat,y_test_cat, verbose = 1)
print("Test loss:", score[0])
print(f"Test Accuracy:{score[1]*100:.2f}%")
```

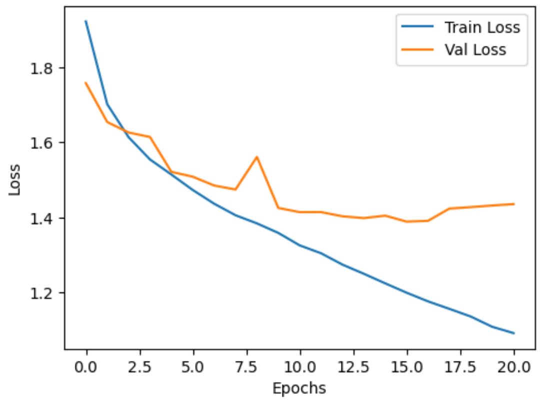
```
313/313              1s 3ms/step -
accuracy: 0.5154 - loss: 1.3739
Test loss: 1.3852113485336304
Test Accuracy:51.97%
```

[31]:

```
plt.figure(figsize=(12,4))

plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
```

```
plt.plot(history.history['val_loss'], label='Val Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```





0.6.8 Model with dropout layers

[32]:
```
model = Sequential()
model.add(Dense(512, activation = 'relu',input_shape = (3072,)))
model.add(Dropout(0.3))
model.add(Dense(256, activation = 'relu'))
model.add(Dropout(0.3))
model.add(Dense(128, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation = 'softmax'))
model.summary()
```

Model: "sequential_6"

Layer (type)	Output Shape	Param #
dense_19 (Dense)	(None, 512)	1,573,376
dropout_2 (Dropout)	(None, 512)	0
dense_20 (Dense)	(None, 256)	131,328
dropout_3 (Dropout)	(None, 256)	0

```
 dense_21 (Dense)                (None, 128)                    32,896

 dropout_4 (Dropout)             (None, 128)                         0

 dense_22 (Dense)                (None, 10)                      1,290
```

Total params: 1,738,890 (6.63 MB)

Trainable params: 1,738,890 (6.63 MB)

Non-trainable params: 0 (0.00 B)

[33]:
```
from keras.callbacks import EarlyStopping

model.compile(loss= 'categorical_crossentropy',optimizer = Adam(learning_rate=0.
 ↪001),metrics = ['accuracy'])

estop = EarlyStopping(monitor = 'val_loss', min_delta = 1e-4, mode = 'min',␣
 ↪patience = 5, verbose = 1, restore_best_weights=True)
history = model.fit(x_subtrain_flat,y_subtrain_cat, batch_size=128, epochs =␣
 ↪500, verbose = 1, validation_data=(x_valid_flat,y_valid_cat),␣
 ↪callbacks=[estop])
```

```
Epoch 1/500
352/352              10s 25ms/step -
accuracy: 0.1807 - loss: 2.2071 - val_accuracy: 0.3238 - val_loss: 1.8934
Epoch 2/500
352/352              8s 23ms/step -
accuracy: 0.2852 - loss: 1.9350 - val_accuracy: 0.3202 - val_loss: 1.8745
Epoch 3/500
352/352              7s 21ms/step -
accuracy: 0.3015 - loss: 1.8916 - val_accuracy: 0.3638 - val_loss: 1.7975
Epoch 4/500
352/352              7s 20ms/step -
accuracy: 0.3119 - loss: 1.8670 - val_accuracy: 0.3634 - val_loss: 1.7696
Epoch 5/500
352/352              8s 23ms/step -
accuracy: 0.3225 - loss: 1.8450 - val_accuracy: 0.3570 - val_loss: 1.7812
Epoch 6/500
352/352              9s 25ms/step -
accuracy: 0.3302 - loss: 1.8304 - val_accuracy: 0.3758 - val_loss: 1.7633
Epoch 7/500
352/352              10s 28ms/step -
accuracy: 0.3340 - loss: 1.8129 - val_accuracy: 0.3810 - val_loss: 1.7565
```

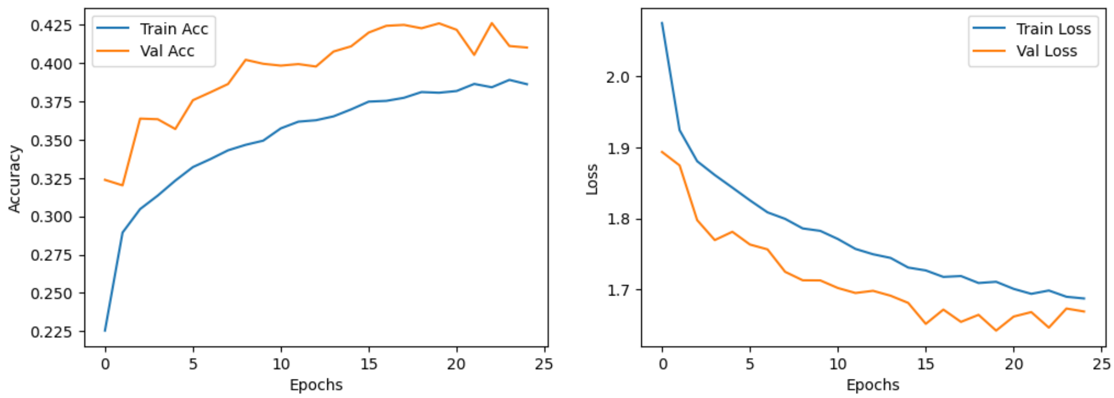
```
Epoch 8/500
352/352            9s 25ms/step -
accuracy: 0.3451 - loss: 1.8041 - val_accuracy: 0.3864 - val_loss: 1.7250
Epoch 9/500
352/352            9s 26ms/step -
accuracy: 0.3491 - loss: 1.7873 - val_accuracy: 0.4022 - val_loss: 1.7131
Epoch 10/500
352/352            8s 22ms/step -
accuracy: 0.3491 - loss: 1.7870 - val_accuracy: 0.3996 - val_loss: 1.7129
Epoch 11/500
352/352            7s 21ms/step -
accuracy: 0.3579 - loss: 1.7663 - val_accuracy: 0.3984 - val_loss: 1.7022
Epoch 12/500
352/352            7s 21ms/step -
accuracy: 0.3630 - loss: 1.7551 - val_accuracy: 0.3994 - val_loss: 1.6952
Epoch 13/500
352/352            7s 21ms/step -
accuracy: 0.3633 - loss: 1.7485 - val_accuracy: 0.3978 - val_loss: 1.6983
Epoch 14/500
352/352            7s 21ms/step -
accuracy: 0.3591 - loss: 1.7584 - val_accuracy: 0.4076 - val_loss: 1.6914
Epoch 15/500
352/352            8s 21ms/step -
accuracy: 0.3694 - loss: 1.7258 - val_accuracy: 0.4110 - val_loss: 1.6813
Epoch 16/500
352/352            8s 21ms/step -
accuracy: 0.3750 - loss: 1.7254 - val_accuracy: 0.4200 - val_loss: 1.6518
Epoch 17/500
352/352            8s 22ms/step -
accuracy: 0.3788 - loss: 1.7118 - val_accuracy: 0.4244 - val_loss: 1.6719
Epoch 18/500
352/352            7s 20ms/step -
accuracy: 0.3794 - loss: 1.7138 - val_accuracy: 0.4250 - val_loss: 1.6546
Epoch 19/500
352/352            7s 21ms/step -
accuracy: 0.3791 - loss: 1.7157 - val_accuracy: 0.4228 - val_loss: 1.6645
Epoch 20/500
352/352            8s 23ms/step -
accuracy: 0.3806 - loss: 1.7142 - val_accuracy: 0.4260 - val_loss: 1.6423
Epoch 21/500
352/352            7s 19ms/step -
accuracy: 0.3810 - loss: 1.7043 - val_accuracy: 0.4218 - val_loss: 1.6621
Epoch 22/500
352/352            7s 19ms/step -
accuracy: 0.3816 - loss: 1.7004 - val_accuracy: 0.4054 - val_loss: 1.6683
Epoch 23/500
352/352            6s 18ms/step -
accuracy: 0.3839 - loss: 1.7022 - val_accuracy: 0.4262 - val_loss: 1.6465
```

```
Epoch 24/500
352/352              6s 18ms/step -
accuracy: 0.3908 - loss: 1.6887 - val_accuracy: 0.4112 - val_loss: 1.6733
Epoch 25/500
352/352              7s 19ms/step -
accuracy: 0.3828 - loss: 1.6960 - val_accuracy: 0.4102 - val_loss: 1.6692
Epoch 25: early stopping
Restoring model weights from the end of the best epoch: 20.
```

[34]:
```python
score = model.evaluate(x_test_flat,y_test_cat, verbose = 1)
print("Test loss:", score[0])
print(f"Test Accuracy:{score[1]*100:.2f}%")
```

```
313/313              1s 3ms/step -
accuracy: 0.4278 - loss: 1.6321
Test loss: 1.6369125843048096
Test Accuracy:42.79%
```

[35]:
```python
plt.figure(figsize=(12,4))

plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```



0.7 Question 3. Next from keras.regularizers import l2

0.7.1 Model building with l2 regularizer

[36]:
```
model = Sequential()
model.add(Dense(512, activation = 'relu', kernel_regularizer=l2(0.
↪0001),input_shape = (3072,)))
model.add(Dense(256, activation = 'relu',kernel_regularizer=l2(0.0001)))
model.add(Dense(128, activation = 'relu',kernel_regularizer=l2(0.005)))
model.add(Dense(10, activation = 'softmax'))
model.summary()
```

Model: "sequential_7"

Layer (type)	Output Shape	Param #
dense_23 (Dense)	(None, 512)	1,573,376
dense_24 (Dense)	(None, 256)	131,328
dense_25 (Dense)	(None, 128)	32,896
dense_26 (Dense)	(None, 10)	1,290

Total params: 1,738,890 (6.63 MB)

Trainable params: 1,738,890 (6.63 MB)

Non-trainable params: 0 (0.00 B)

0.7.2 Without using early stopping

[37]:
```
model.compile(loss= 'categorical_crossentropy',optimizer = Adam(learning_rate=0.
↪001),metrics = ['accuracy'])

estop = EarlyStopping(monitor = 'val_loss', min_delta = 1e-4, mode = 'min',␣
↪patience = 5, verbose = 1, restore_best_weights=True)
history = model.fit(x_subtrain_flat,y_subtrain_cat, batch_size=128, epochs =␣
↪500, verbose = 1, validation_data=(x_valid_flat,y_valid_cat),␣
↪callbacks=[estop])
```

```
Epoch 1/500
352/352              10s 23ms/step -
accuracy: 0.2553 - loss: 2.6025 - val_accuracy: 0.3716 - val_loss: 1.9682
Epoch 2/500
352/352              7s 20ms/step -
accuracy: 0.3771 - loss: 1.9061 - val_accuracy: 0.4190 - val_loss: 1.7790
Epoch 3/500
352/352              7s 20ms/step -
accuracy: 0.4159 - loss: 1.7583 - val_accuracy: 0.4314 - val_loss: 1.7220
Epoch 4/500
352/352              7s 20ms/step -
accuracy: 0.4350 - loss: 1.6784 - val_accuracy: 0.4676 - val_loss: 1.6170
Epoch 5/500
352/352              7s 20ms/step -
accuracy: 0.4565 - loss: 1.6113 - val_accuracy: 0.4622 - val_loss: 1.6180
Epoch 6/500
352/352              7s 20ms/step -
accuracy: 0.4679 - loss: 1.5756 - val_accuracy: 0.4544 - val_loss: 1.6137
Epoch 7/500
352/352              7s 20ms/step -
accuracy: 0.4787 - loss: 1.5499 - val_accuracy: 0.4600 - val_loss: 1.5985
Epoch 8/500
352/352              7s 21ms/step -
accuracy: 0.4832 - loss: 1.5293 - val_accuracy: 0.4656 - val_loss: 1.5763
Epoch 9/500
352/352              7s 20ms/step -
accuracy: 0.4918 - loss: 1.5201 - val_accuracy: 0.4738 - val_loss: 1.5532
Epoch 10/500
352/352              7s 20ms/step -
accuracy: 0.5001 - loss: 1.4908 - val_accuracy: 0.4738 - val_loss: 1.5866
Epoch 11/500
352/352              7s 20ms/step -
accuracy: 0.5093 - loss: 1.4653 - val_accuracy: 0.4852 - val_loss: 1.5406
Epoch 12/500
352/352              7s 20ms/step -
accuracy: 0.5160 - loss: 1.4462 - val_accuracy: 0.4908 - val_loss: 1.5158
Epoch 13/500
352/352              7s 20ms/step -
accuracy: 0.5194 - loss: 1.4355 - val_accuracy: 0.4954 - val_loss: 1.5275
Epoch 14/500
352/352              7s 21ms/step -
accuracy: 0.5283 - loss: 1.4164 - val_accuracy: 0.5062 - val_loss: 1.4899
Epoch 15/500
352/352              7s 20ms/step -
accuracy: 0.5300 - loss: 1.4028 - val_accuracy: 0.4844 - val_loss: 1.5423
Epoch 16/500
352/352              7s 20ms/step -
accuracy: 0.5437 - loss: 1.3788 - val_accuracy: 0.5064 - val_loss: 1.4921
```

```
Epoch 17/500
352/352              7s 20ms/step -
accuracy: 0.5483 - loss: 1.3751 - val_accuracy: 0.5058 - val_loss: 1.4971
Epoch 18/500
352/352              7s 20ms/step -
accuracy: 0.5472 - loss: 1.3703 - val_accuracy: 0.5154 - val_loss: 1.4749
Epoch 19/500
352/352              8s 21ms/step -
accuracy: 0.5584 - loss: 1.3410 - val_accuracy: 0.4946 - val_loss: 1.5050
Epoch 20/500
352/352              7s 21ms/step -
accuracy: 0.5597 - loss: 1.3463 - val_accuracy: 0.5168 - val_loss: 1.4709
Epoch 21/500
352/352              7s 20ms/step -
accuracy: 0.5629 - loss: 1.3367 - val_accuracy: 0.5042 - val_loss: 1.4976
Epoch 22/500
352/352              7s 20ms/step -
accuracy: 0.5648 - loss: 1.3345 - val_accuracy: 0.5042 - val_loss: 1.4977
Epoch 23/500
352/352              7s 21ms/step -
accuracy: 0.5659 - loss: 1.3266 - val_accuracy: 0.5152 - val_loss: 1.4802
Epoch 24/500
352/352              7s 21ms/step -
accuracy: 0.5714 - loss: 1.3183 - val_accuracy: 0.5066 - val_loss: 1.5034
Epoch 25/500
352/352              7s 20ms/step -
accuracy: 0.5798 - loss: 1.2953 - val_accuracy: 0.5174 - val_loss: 1.5022
Epoch 25: early stopping
Restoring model weights from the end of the best epoch: 20.
```

[38]:

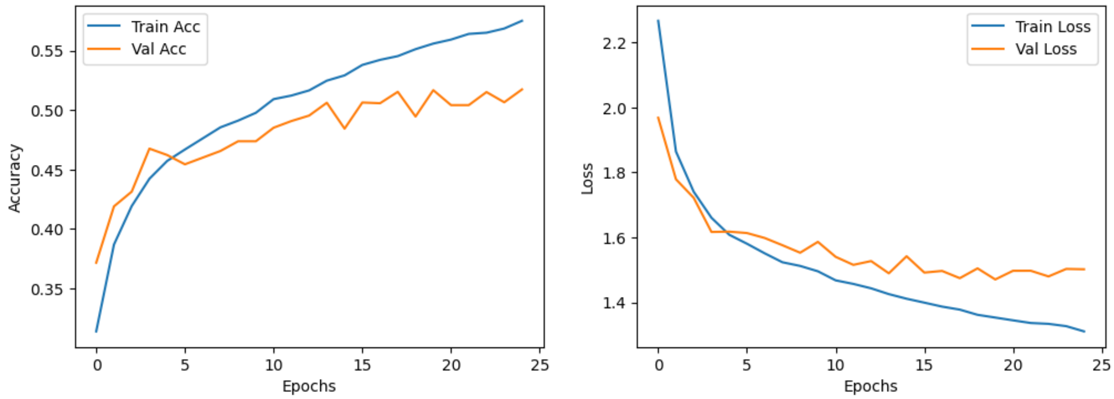
```
score = model.evaluate(x_test_flat,y_test_cat, verbose = 1)
print("Test loss:", score[0])
print(f"Test Accuracy:{score[1]*100:.2f}%")
```

```
313/313              1s 4ms/step -
accuracy: 0.5204 - loss: 1.4501
Test loss: 1.4567960500717163
Test Accuracy:51.61%
```

[39]:

```
plt.figure(figsize=(12,4))

plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

```
plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```



0.7.3 using early stopping along with l2 regularization

[40]:
```
estop = EarlyStopping(monitor = 'val_loss', min_delta = 1e-4, mode = 'min',␣
 ↪patience = 5, verbose = 1, restore_best_weights=True)
history = model.fit(x_subtrain_flat,y_subtrain_cat, batch_size=128, epochs =␣
 ↪500, verbose = 1, validation_data=(x_valid_flat,y_valid_cat),␣
 ↪callbacks=[estop])
```

```
Epoch 1/500
352/352              9s 24ms/step -
accuracy: 0.5652 - loss: 1.3301 - val_accuracy: 0.4984 - val_loss: 1.5603
Epoch 2/500
352/352              7s 21ms/step -
accuracy: 0.5633 - loss: 1.3336 - val_accuracy: 0.5030 - val_loss: 1.5436
Epoch 3/500
352/352              7s 20ms/step -
accuracy: 0.5672 - loss: 1.3255 - val_accuracy: 0.5118 - val_loss: 1.4963
Epoch 4/500
352/352              7s 21ms/step -
accuracy: 0.5742 - loss: 1.3156 - val_accuracy: 0.4996 - val_loss: 1.5386
Epoch 5/500
352/352              7s 20ms/step -
accuracy: 0.5724 - loss: 1.3093 - val_accuracy: 0.5086 - val_loss: 1.5058
Epoch 6/500
```

```
352/352          8s 21ms/step -
accuracy: 0.5819 - loss: 1.2856 - val_accuracy: 0.5030 - val_loss: 1.5206
Epoch 7/500
352/352          7s 20ms/step -
accuracy: 0.5764 - loss: 1.2972 - val_accuracy: 0.5168 - val_loss: 1.5215
Epoch 8/500
352/352          7s 20ms/step -
accuracy: 0.5907 - loss: 1.2719 - val_accuracy: 0.5230 - val_loss: 1.4814
Epoch 9/500
352/352          7s 20ms/step -
accuracy: 0.5892 - loss: 1.2758 - val_accuracy: 0.5266 - val_loss: 1.4918
Epoch 10/500
352/352          7s 21ms/step -
accuracy: 0.5846 - loss: 1.2767 - val_accuracy: 0.5162 - val_loss: 1.4942
Epoch 11/500
352/352          7s 20ms/step -
accuracy: 0.5923 - loss: 1.2511 - val_accuracy: 0.5014 - val_loss: 1.5157
Epoch 12/500
352/352          7s 20ms/step -
accuracy: 0.5935 - loss: 1.2714 - val_accuracy: 0.5276 - val_loss: 1.5037
Epoch 13/500
352/352          7s 20ms/step -
accuracy: 0.5961 - loss: 1.2632 - val_accuracy: 0.5250 - val_loss: 1.4913
Epoch 13: early stopping
Restoring model weights from the end of the best epoch: 8.
```

[41]:

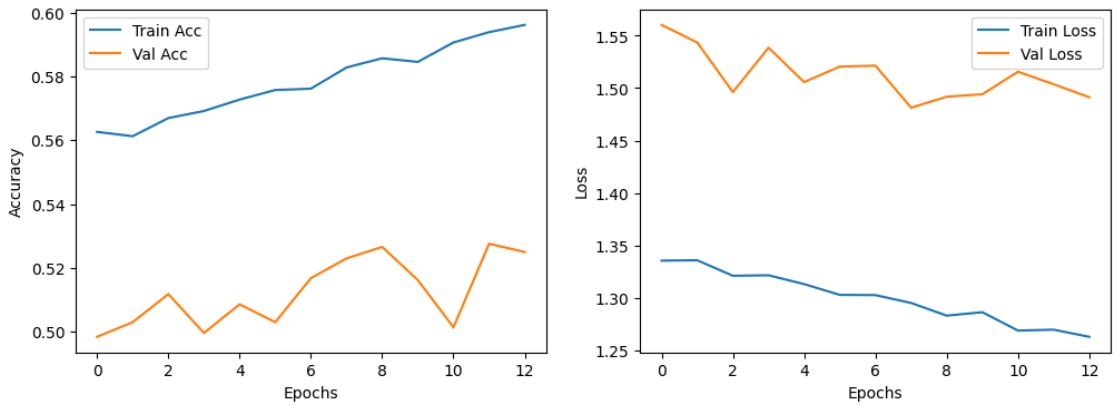
```
score = model.evaluate(x_test_flat,y_test_cat, verbose = 1)
print("Test loss:", score[0])
print(f"Test Accuracy:{score[1]*100:.2f}%")
```

```
313/313          1s 3ms/step -
accuracy: 0.5239 - loss: 1.4569
Test loss: 1.465133547782898
Test Accuracy:52.07%
```

[42]:

```
plt.figure(figsize=(12,4))

plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.xlabel('Epochs')
```

```
plt.ylabel('Loss')
plt.legend()

plt.show()
```



0.8 Question 4: Now, let's see how we can proceed to do perform some hyperparameter tuning and find out the appropriate parameter value. The following part is done for a very simple model with one hidden layer and an output layer. The number of neurons and the dropout parameter is being tuned to find appropriate ones.

[43]:
```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.utils import to_categorical
from keras.optimizers import SGD,Adam
import keras_tuner as kt
```

[44]:
```
(x_train,y_train),(x_test,y_test) = mnist.load_data()
print(x_train.shape)

x_train = x_train.reshape(-1,28*28).astype('float32')/255.0
x_test = x_test.reshape(-1,28*28).astype('float32')/255.0

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

```
(60000, 28, 28)
(60000, 784)
(10000, 784)
```

[45]:
```
y_train_ = to_categorical(y_train,10)
y_test_ = to_categorical(y_test,10)

def build_model(hp):
    model = Sequential()
    model.add(Flatten(input_shape= (28*28,)))

    units = hp.Int('units', min_value = 64, max_value = 512, step = 64)
    model.add(Dense(units,activation = 'relu'))
    dropout_rate = hp.Float('dropout',min_value = 0.0, max_value = 0.5, step =␣
 ↪0.1)
    model.add(Dropout(dropout_rate))
    model.add(Dense(10, activation = 'softmax'))

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

[46]:
```
tuner = kt.RandomSearch(
    build_model,
    objective = 'val_accuracy',
    max_trials = 10,
    executions_per_trial = 1,
    directory = 'mnist_tuning',
    project_name = 'dense_dropout_tune',
    overwrite=True
)

tuner.search(x_train,y_train_, epochs = 10, validation_split = 0.2,batch_size =␣
 ↪128,callbacks = [keras.callbacks.EarlyStopping(monitor = 'val_loss',␣
 ↪patience = 5)])
best_model = tuner.get_best_models(num_models = 1)[0]

test_loss, test_acc = best_model.evaluate(x_test,y_test_)
print("Test Accuracy:",test_acc)

best_hps = tuner.get_best_hyperparameters(1)[0]
print("Best Units:", best_hps.get('units'))
print("Best dropout:",best_hps.get('dropout'))
```

```
Trial 10 Complete [00h 00m 15s]
val_accuracy: 0.9211666584014893

Best val_accuracy So Far: 0.9235833287239075
Total elapsed time: 00h 02m 53s
```

```
313/313              1s 2ms/step -
accuracy: 0.9105 - loss: 0.3241
Test Accuracy: 0.9232000112533569
Best Units: 320
Best dropout: 0.2
```

[47]:
```python
best_model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 320)	251,200
dropout (Dropout)	(None, 320)	0
dense_1 (Dense)	(None, 10)	3,210

```
Total params: 254,410 (993.79 KB)

Trainable params: 254,410 (993.79 KB)

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

[48]:
```python
history = best_model.fit(x_train,y_train_, batch_size=128, epochs = 50, verbose
    = 1, validation_split=0.2,callbacks = [keras.callbacks.EarlyStopping(monitor
    = 'val_loss', patience = 5)])
```

```
Epoch 1/50
375/375              2s 4ms/step -
accuracy: 0.9090 - loss: 0.3180 - val_accuracy: 0.9255 - val_loss: 0.2722
Epoch 2/50
375/375              2s 4ms/step -
accuracy: 0.9152 - loss: 0.3035 - val_accuracy: 0.9270 - val_loss: 0.2646
Epoch 3/50
375/375              2s 5ms/step -
accuracy: 0.9161 - loss: 0.2941 - val_accuracy: 0.9298 - val_loss: 0.2577
Epoch 4/50
375/375              2s 4ms/step -
accuracy: 0.9172 - loss: 0.2921 - val_accuracy: 0.9303 - val_loss: 0.2517
Epoch 5/50
375/375              2s 5ms/step -
```

```
accuracy: 0.9197 - loss: 0.2824 - val_accuracy: 0.9316 - val_loss: 0.2458
Epoch 6/50
375/375            2s 5ms/step -
accuracy: 0.9223 - loss: 0.2770 - val_accuracy: 0.9337 - val_loss: 0.2405
Epoch 7/50
375/375            3s 7ms/step -
accuracy: 0.9268 - loss: 0.2656 - val_accuracy: 0.9352 - val_loss: 0.2350
Epoch 8/50
375/375            2s 5ms/step -
accuracy: 0.9239 - loss: 0.2645 - val_accuracy: 0.9362 - val_loss: 0.2305
Epoch 9/50
375/375            2s 4ms/step -
accuracy: 0.9281 - loss: 0.2586 - val_accuracy: 0.9377 - val_loss: 0.2257
Epoch 10/50
375/375            1s 4ms/step -
accuracy: 0.9309 - loss: 0.2508 - val_accuracy: 0.9388 - val_loss: 0.2217
Epoch 11/50
375/375            1s 4ms/step -
accuracy: 0.9322 - loss: 0.2432 - val_accuracy: 0.9400 - val_loss: 0.2173
Epoch 12/50
375/375            2s 4ms/step -
accuracy: 0.9310 - loss: 0.2428 - val_accuracy: 0.9411 - val_loss: 0.2133
Epoch 13/50
375/375            1s 4ms/step -
accuracy: 0.9351 - loss: 0.2337 - val_accuracy: 0.9423 - val_loss: 0.2098
Epoch 14/50
375/375            1s 4ms/step -
accuracy: 0.9338 - loss: 0.2364 - val_accuracy: 0.9433 - val_loss: 0.2059
Epoch 15/50
375/375            1s 4ms/step -
accuracy: 0.9344 - loss: 0.2321 - val_accuracy: 0.9444 - val_loss: 0.2024
Epoch 16/50
375/375            2s 5ms/step -
accuracy: 0.9367 - loss: 0.2207 - val_accuracy: 0.9447 - val_loss: 0.1991
Epoch 17/50
375/375            1s 4ms/step -
accuracy: 0.9358 - loss: 0.2241 - val_accuracy: 0.9456 - val_loss: 0.1965
Epoch 18/50
375/375            2s 4ms/step -
accuracy: 0.9386 - loss: 0.2182 - val_accuracy: 0.9463 - val_loss: 0.1933
Epoch 19/50
375/375            2s 4ms/step -
accuracy: 0.9408 - loss: 0.2071 - val_accuracy: 0.9473 - val_loss: 0.1903
Epoch 20/50
375/375            2s 4ms/step -
accuracy: 0.9416 - loss: 0.2077 - val_accuracy: 0.9477 - val_loss: 0.1876
Epoch 21/50
375/375            2s 4ms/step -
```

```
accuracy: 0.9411 - loss: 0.2061 - val_accuracy: 0.9483 - val_loss: 0.1847
Epoch 22/50
375/375            2s 5ms/step -
accuracy: 0.9429 - loss: 0.2023 - val_accuracy: 0.9499 - val_loss: 0.1823
Epoch 23/50
375/375            2s 4ms/step -
accuracy: 0.9432 - loss: 0.2012 - val_accuracy: 0.9496 - val_loss: 0.1801
Epoch 24/50
375/375            2s 5ms/step -
accuracy: 0.9447 - loss: 0.1935 - val_accuracy: 0.9504 - val_loss: 0.1776
Epoch 25/50
375/375            2s 6ms/step -
accuracy: 0.9476 - loss: 0.1912 - val_accuracy: 0.9525 - val_loss: 0.1749
Epoch 26/50
375/375            3s 8ms/step -
accuracy: 0.9465 - loss: 0.1877 - val_accuracy: 0.9523 - val_loss: 0.1728
Epoch 27/50
375/375            3s 8ms/step -
accuracy: 0.9474 - loss: 0.1862 - val_accuracy: 0.9531 - val_loss: 0.1706
Epoch 28/50
375/375            3s 7ms/step -
accuracy: 0.9468 - loss: 0.1876 - val_accuracy: 0.9534 - val_loss: 0.1686
Epoch 29/50
375/375            2s 6ms/step -
accuracy: 0.9485 - loss: 0.1836 - val_accuracy: 0.9541 - val_loss: 0.1666
Epoch 30/50
375/375            2s 6ms/step -
accuracy: 0.9483 - loss: 0.1779 - val_accuracy: 0.9544 - val_loss: 0.1647
Epoch 31/50
375/375            3s 7ms/step -
accuracy: 0.9482 - loss: 0.1792 - val_accuracy: 0.9548 - val_loss: 0.1627
Epoch 32/50
375/375            2s 5ms/step -
accuracy: 0.9515 - loss: 0.1737 - val_accuracy: 0.9553 - val_loss: 0.1609
Epoch 33/50
375/375            2s 5ms/step -
accuracy: 0.9491 - loss: 0.1751 - val_accuracy: 0.9557 - val_loss: 0.1591
Epoch 34/50
375/375            2s 4ms/step -
accuracy: 0.9526 - loss: 0.1663 - val_accuracy: 0.9565 - val_loss: 0.1576
Epoch 35/50
375/375            2s 5ms/step -
accuracy: 0.9514 - loss: 0.1672 - val_accuracy: 0.9573 - val_loss: 0.1557
Epoch 36/50
375/375            2s 5ms/step -
accuracy: 0.9518 - loss: 0.1689 - val_accuracy: 0.9575 - val_loss: 0.1543
Epoch 37/50
375/375            2s 5ms/step -
```

```
accuracy: 0.9562 - loss: 0.1581 - val_accuracy: 0.9582 - val_loss: 0.1526
Epoch 38/50
375/375              2s 5ms/step -
accuracy: 0.9558 - loss: 0.1584 - val_accuracy: 0.9587 - val_loss: 0.1512
Epoch 39/50
375/375              2s 4ms/step -
accuracy: 0.9537 - loss: 0.1617 - val_accuracy: 0.9587 - val_loss: 0.1498
Epoch 40/50
375/375              2s 5ms/step -
accuracy: 0.9546 - loss: 0.1581 - val_accuracy: 0.9592 - val_loss: 0.1482
Epoch 41/50
375/375              2s 4ms/step -
accuracy: 0.9554 - loss: 0.1565 - val_accuracy: 0.9603 - val_loss: 0.1468
Epoch 42/50
375/375              2s 4ms/step -
accuracy: 0.9557 - loss: 0.1566 - val_accuracy: 0.9601 - val_loss: 0.1455
Epoch 43/50
375/375              2s 4ms/step -
accuracy: 0.9568 - loss: 0.1529 - val_accuracy: 0.9599 - val_loss: 0.1441
Epoch 44/50
375/375              2s 4ms/step -
accuracy: 0.9573 - loss: 0.1486 - val_accuracy: 0.9609 - val_loss: 0.1430
Epoch 45/50
375/375              2s 5ms/step -
accuracy: 0.9573 - loss: 0.1520 - val_accuracy: 0.9606 - val_loss: 0.1419
Epoch 46/50
375/375              2s 4ms/step -
accuracy: 0.9574 - loss: 0.1528 - val_accuracy: 0.9610 - val_loss: 0.1405
Epoch 47/50
375/375              2s 5ms/step -
accuracy: 0.9596 - loss: 0.1436 - val_accuracy: 0.9613 - val_loss: 0.1396
Epoch 48/50
375/375              2s 4ms/step -
accuracy: 0.9585 - loss: 0.1478 - val_accuracy: 0.9613 - val_loss: 0.1382
Epoch 49/50
375/375              2s 4ms/step -
accuracy: 0.9593 - loss: 0.1450 - val_accuracy: 0.9617 - val_loss: 0.1371
Epoch 50/50
375/375              2s 6ms/step -
accuracy: 0.9593 - loss: 0.1439 - val_accuracy: 0.9618 - val_loss: 0.1360
```

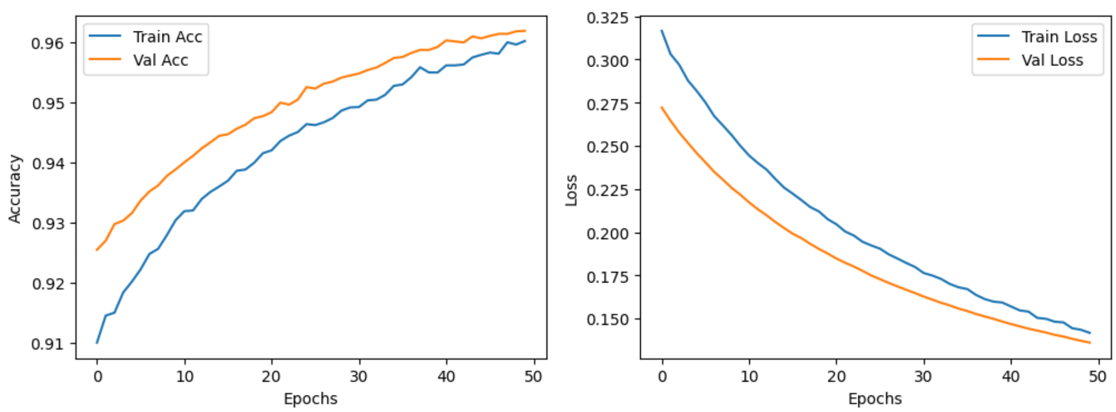
[49]:

```
score = best_model.evaluate(x_test,y_test_, verbose = 1)
print("Test loss:", score[0])
print(f"Test Accuracy:{score[1]*100:.2f}%")
```

```
313/313              2s 6ms/step -
accuracy: 0.9556 - loss: 0.1555
Test loss: 0.13252055644989014
```

Test Accuracy:96.27%

[50]:
```
plt.figure(figsize=(12,4))

plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```



0.8.1 Tuning further parameters like

- Number of hidden layers
- units per layers
- dropout per layer
- optimizer type
- learning rate

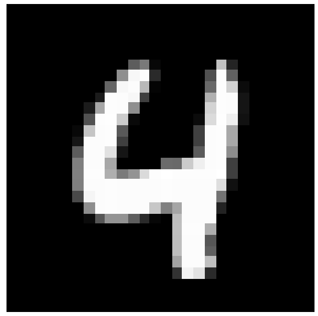
[51]:
```
from tensorflow import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Flatten, Dropout
from keras.utils import to_categorical
```

```
import keras_tuner as kt
from keras.regularizers import l2
import numpy as np
```

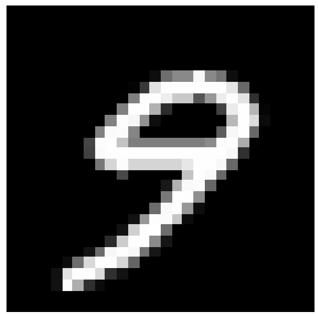
0.8.2 Loading and preprocessing the dataset

[52]:
```
(x_train,y_train),(x_test,y_test) = mnist.load_data()

## Plotting some random images from the dataset
random_indices = np.random.choice(len(x_train), size=5, replace=False)
plt.figure(figsize=(2, 10))
for i, idx in enumerate(random_indices):
    plt.subplot(5, 1, i+1)
    plt.imshow(x_train[idx],cmap = 'gray')
    plt.title(int(y_train[idx]))
    plt.axis('off')

plt.tight_layout()
plt.show()

x_train = x_train.reshape(-1,28*28).astype('float32')/255.0
x_test = x_test.reshape(-1,28*28).astype('float32')/255.0
y_train_ = to_categorical(y_train,10)
y_test_ = to_categorical(y_test, 10)
```

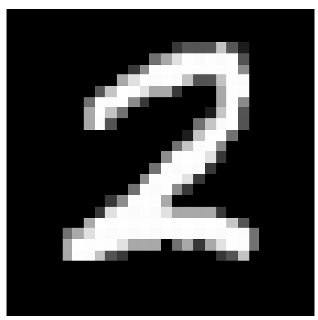
4



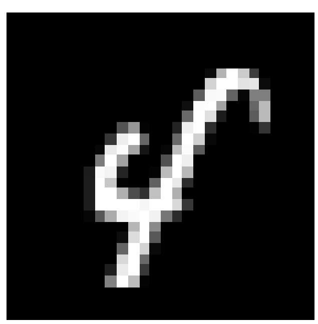
9



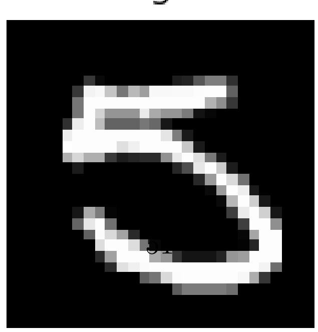
2



4



5



0.8.3 Building model for tuner

- with L2 regularization

[53]:
```
def build_model(hp):
    model = Sequential()
    model.add(Flatten(input_shape = (784,)))

    ## Tuning the number of layers
    for i in range(hp.Int('num_layers',1,3)):
        model.add(Dense(
            ## no of nodes in each layer
            units = hp.Int(f'units_{i}', min_value = 64, max_value = 512, step␣
↪= 64),
            activation = 'relu',
            kernel_regularizer=l2(
                hp.Choice(f'l2_{i}', values = [0.0,1e-4,1e-3,1e-2])
            )
        ))
        ## Tuning the dropout rate
        model.add(Dropout(
            rate = hp.Float(f'dropout_{i}', min_value = 0.0,max_value = 0.5,␣
↪step = 0.05)
        ))

    ## output layer
    model.add(Dense(10, activation = 'softmax'))

    ## Tuning the optimizer type and the learning rate
    optimizer_choice = hp.Choice('optimizer', values = ['adam','sgd'])
    learning_rate = hp.Choice('learning_rate', values=[1e-2,1e-3,1e-4])

    if optimizer_choice == 'adam':
        optimizer = keras.optimizers.Adam(learning_rate=learning_rate)
    else:
        optimizer = keras.optimizers.SGD(learning_rate=learning_rate)

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

[54]:
```python
tuner = kt.RandomSearch(
    build_model,
    objective = 'val_accuracy',
    max_trials=20,
    executions_per_trial = 2,
    overwrite = True,
    directory = 'mnist_tuning',
    project_name = 'advanced_dense_tune'
)

tuner.search(
    x_train,y_train_,
    epochs = 10,
    validation_split = 0.2,
    batch_size = 128,
    callbacks = [keras.callbacks.EarlyStopping(monitor = 'val_loss', patience =␣
↪3,min_delta=1e-4, restore_best_weights = True)]
)
```

```
Trial 20 Complete [00h 00m 30s]
val_accuracy: 0.28724999725818634

Best val_accuracy So Far: 0.9781250059604645
Total elapsed time: 00h 16m 35s
```

[55]:
```python
# best model and hyperparameters
best_model = tuner.get_best_models(num_models=1)[0]
best_hps = tuner.get_best_hyperparameters(1)[0]
print("Best Hyperparameters:")
for key in best_hps.values.keys():
    print(f"{key}: {best_hps.get(key)}")

# Final training
history = best_model.fit(
    x_train, y_train_,
    batch_size=128,
    epochs=50,
    validation_split=0.2,
    callbacks=[keras.callbacks.EarlyStopping(monitor='val_loss', patience =␣
↪3,min_delta=1e-4, restore_best_weights = True)]
)

# Evaluate
score = best_model.evaluate(x_test, y_test_, verbose=1)
print("Test loss:", score[0])
print(f"Test Accuracy: {score[1]*100:.2f}%")
```

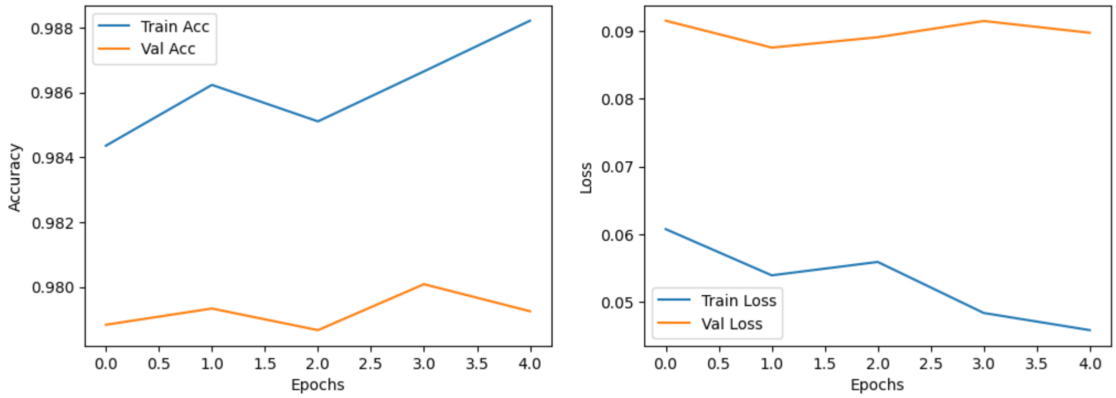
```
Best Hyperparameters:
```

```
num_layers: 3
units_0: 256
l2_0: 0.0
dropout_0: 0.30000000000000004
optimizer: adam
learning_rate: 0.001
units_1: 512
l2_1: 0.0
dropout_1: 0.1
units_2: 448
l2_2: 0.0001
dropout_2: 0.0
Epoch 1/50
375/375              5s 10ms/step -
accuracy: 0.9851 - loss: 0.0591 - val_accuracy: 0.9788 - val_loss: 0.0916
Epoch 2/50
375/375              3s 9ms/step -
accuracy: 0.9868 - loss: 0.0522 - val_accuracy: 0.9793 - val_loss: 0.0876
Epoch 3/50
375/375              3s 9ms/step -
accuracy: 0.9856 - loss: 0.0532 - val_accuracy: 0.9787 - val_loss: 0.0891
Epoch 4/50
375/375              4s 10ms/step -
accuracy: 0.9869 - loss: 0.0471 - val_accuracy: 0.9801 - val_loss: 0.0915
Epoch 5/50
375/375              3s 9ms/step -
accuracy: 0.9880 - loss: 0.0465 - val_accuracy: 0.9793 - val_loss: 0.0898
313/313              1s 3ms/step -
accuracy: 0.9778 - loss: 0.0878
Test loss: 0.07718071341514587
Test Accuracy: 98.14%
```

[56]:

```
plt.figure(figsize=(12,4))

plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

```
plt.show()
```



0.9 Then later go back to the cifar10 dataset problem and come up with your best model

0.9.1 hyper parameter tuning for cifar 10 dataset

0.9.2 importing the necessary libraries

[57]:
```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow import keras
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Input
from keras.regularizers import l2
from keras.utils import to_categorical
import keras_tuner as kt
```

0.9.3 loading the cifar10 dataset

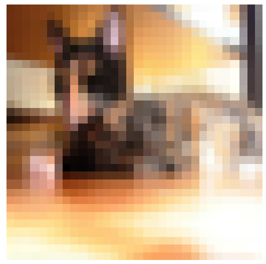
[58]:
```
(x_train,y_train),(x_test,y_test) = cifar10.load_data()

# class label mapping
label_map = {
    0: 'airplane', 1: 'automobile', 2: 'bird', 3: 'cat', 4: 'deer',
    5: 'dog', 6: 'frog', 7: 'horse', 8: 'ship', 9: 'truck'
}
```

0.9.4 Showing some sample images

[59]:
```
idx = np.random.choice(len(x_train), 5, replace = False)
plt.figure(figsize = (2,10))
for i, id in enumerate(idx):
    plt.subplot(5,1,i+1)
    plt.imshow(x_train[id])
    plt.title(label_map[int(y_train[id])])
    plt.axis('off')
plt.show()
```

dog

cat



truck

truck

horse

0.9.5 preprocessing the dataset

[60]:
```python
## normalizing
x_train = x_train.astype('float32')/255.0
x_test = x_test.astype('float32')/255.0
x_train = x_train.reshape(-1,32*32*3)
x_test = x_test.reshape(-1,32*32*3)

y_train_ = to_categorical(y_train,10)
y_test_ = to_categorical(y_test,10)
```

0.9.6 model building for hyperparameter tuning

[61]:
```python
def build_model(hp):
    model = Sequential()
    model.add(Input(shape=(32*32*3,)))
    # Tune number of hidden layers (1-3)
    for i in range(hp.Int('num_layers', 1, 3)):
        model.add(Dense(
            units=hp.Int(f'units_{i}', min_value=128, max_value=512, step=64),
            activation='relu',
            kernel_regularizer=l2(hp.Choice(f'l2_{i}', values=[0.0, 1e-4,␣
↪1e-3]))
        ))
        model.add(Dropout(
            rate=hp.Float(f'dropout_{i}', min_value=0.0, max_value=0.5, step=0.
↪1)
        ))

    # Output layer
    model.add(Dense(10, activation='softmax'))

    # Tune optimizer type & learning rate
    optimizer_choice = hp.Choice('optimizer', values=['adam', 'sgd'])
    learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])

    if optimizer_choice == 'adam':
        optimizer = keras.optimizers.Adam(learning_rate=learning_rate)
    else:
        optimizer = keras.optimizers.SGD(learning_rate=learning_rate,␣
↪momentum=0.9)

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

```
    return model
```

[62]:
```
tuner = kt.RandomSearch(
    build_model,
    objective='val_accuracy',
    max_trials=10,
    executions_per_trial=1,
    overwrite=True,
    directory='cifar10_tuning',
    project_name='dense_dropout_l2_tune'
)

tuner.search(
    x_train, y_train_,
    epochs=20,
    validation_split=0.2,
    batch_size=128,
    callbacks=[keras.callbacks.EarlyStopping(monitor='val_loss', patience =␣
↪3,min_delta=1e-4, restore_best_weights = True)]
)
```

```
Trial 10 Complete [00h 01m 22s]
val_accuracy: 0.20059999823570251

Best val_accuracy So Far: 0.5169000029563904
Total elapsed time: 00h 12m 32s
```

0.9.7 Best model and hyperparameters

[63]:
```
best_model = tuner.get_best_models(num_models=1)[0]
best_hps = tuner.get_best_hyperparameters(1)[0]
print("\nBest Hyperparameters:")
for key in best_hps.values.keys():
    print(f"{key}: {best_hps.get(key)}")
```

```
Best Hyperparameters:
num_layers: 3
units_0: 384
l2_0: 0.0001
dropout_0: 0.1
optimizer: sgd
learning_rate: 0.01
units_1: 320
l2_1: 0.001
dropout_1: 0.1
units_2: 128
l2_2: 0.0
```

dropout_2: 0.0

0.9.8 Training the best model

[64]:
```
history = best_model.fit(
    x_train, y_train_,
    batch_size=128,
    epochs=50,
    validation_split=0.2,
    callbacks=[keras.callbacks.EarlyStopping(monitor='val_loss', patience =
 ↪3,min_delta=1e-4, restore_best_weights = True)]
)
```

```
Epoch 1/50
313/313              8s 20ms/step -
accuracy: 0.5608 - loss: 1.3661 - val_accuracy: 0.4981 - val_loss: 1.5471
Epoch 2/50
313/313              5s 16ms/step -
accuracy: 0.5558 - loss: 1.3684 - val_accuracy: 0.5067 - val_loss: 1.5259
Epoch 3/50
313/313              6s 18ms/step -
accuracy: 0.5577 - loss: 1.3547 - val_accuracy: 0.5179 - val_loss: 1.4900
Epoch 4/50
313/313              5s 17ms/step -
accuracy: 0.5693 - loss: 1.3283 - val_accuracy: 0.5103 - val_loss: 1.5069
Epoch 5/50
313/313              6s 18ms/step -
accuracy: 0.5712 - loss: 1.3270 - val_accuracy: 0.5167 - val_loss: 1.4978
Epoch 6/50
313/313              5s 17ms/step -
accuracy: 0.5786 - loss: 1.3023 - val_accuracy: 0.5223 - val_loss: 1.4850
Epoch 7/50
313/313              6s 18ms/step -
accuracy: 0.5784 - loss: 1.3036 - val_accuracy: 0.5225 - val_loss: 1.5013
Epoch 8/50
313/313              5s 16ms/step -
accuracy: 0.5848 - loss: 1.2859 - val_accuracy: 0.5355 - val_loss: 1.4660
Epoch 9/50
313/313              5s 15ms/step -
accuracy: 0.5829 - loss: 1.2819 - val_accuracy: 0.5328 - val_loss: 1.4694
Epoch 10/50
313/313              4s 14ms/step -
accuracy: 0.5939 - loss: 1.2719 - val_accuracy: 0.5330 - val_loss: 1.4760
Epoch 11/50
313/313              4s 14ms/step -
accuracy: 0.5964 - loss: 1.2596 - val_accuracy: 0.5281 - val_loss: 1.4908
```

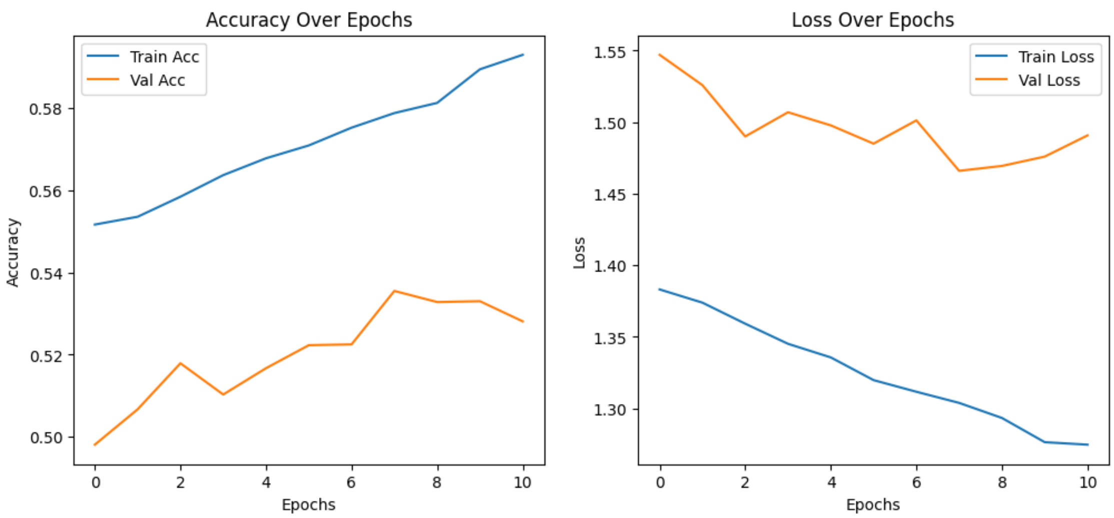
0.9.9 Evaluating the model

[65]:
```
score = best_model.evaluate(x_test, y_test_, verbose=1)
print("Test loss:", score[0])
print(f"Test Accuracy: {score[1]*100:.2f}%")
```

```
313/313              1s 3ms/step -
accuracy: 0.5398 - loss: 1.4360
Test loss: 1.4406636953353882
Test Accuracy: 53.62%
```

0.9.10 Accuracy and loss plot

[66]:
```
plt.figure(figsize=(12, 5))

# Accuracy plot
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Accuracy Over Epochs')

# Loss plot
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Loss Over Epochs')

plt.show()
```


Accuracy Over Epochs
Train Acc
Val Acc
Accuracy
0.58
0.56
0.54
0.52
0.50
0
2
4
6
8
10
Epochs
Loss Over Epochs
Train Loss
Val Loss
Loss
1.55
1.50
1.45
1.40
1.35
1.30
0
2
4
6
8
10
Epochs