Cifar 100

* tackling more on the fine labels

By:Joel Poah

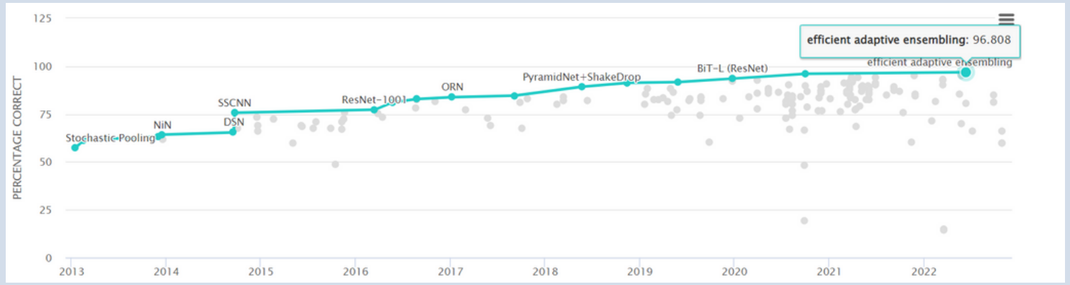
Admin No:P2112729

Class:DAAA/FT/2B/06

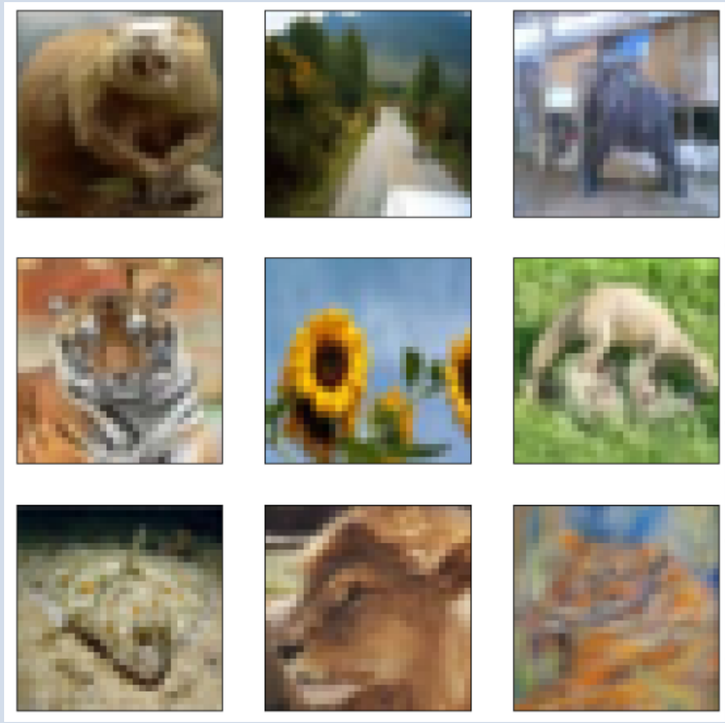
EDA + research

SuperClass	Classes
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles_1	bicycle, bus, motorcycle, pickup truck, train
vehicles_2	lawn-mower, rocket, streetcar, tank, tractor

- Cifar 100 contains 60 thousand tiny dataset images from tf.keras . They are colored which means they have 3 channels and each channel has 32 by 32 pixels
- Collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.



- State-of-the-art efficient net
- test acc 96.908%



```
y_test_labels = y_test
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_val = to_categorical(y_val)
```

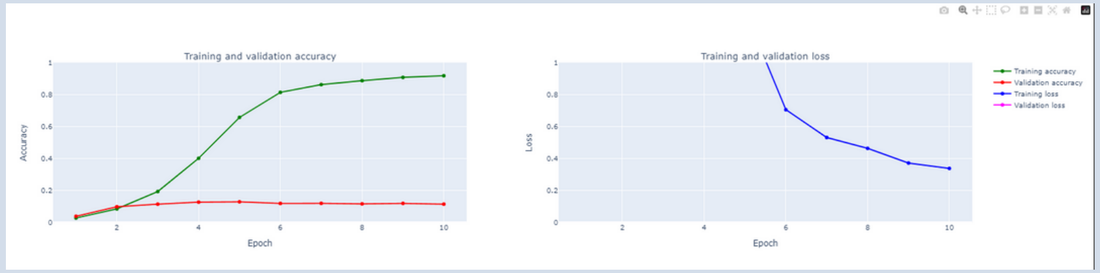
- images are generally centered
- Split into train, validation and test
- Stratified splitting used, equal number of data points per class
- Data is extremely scarce only 400 per calss
- pixel normalization was chosen instead of centering/standardization techniques for faster learning/ convergence in neural networks
- one hot encoding used

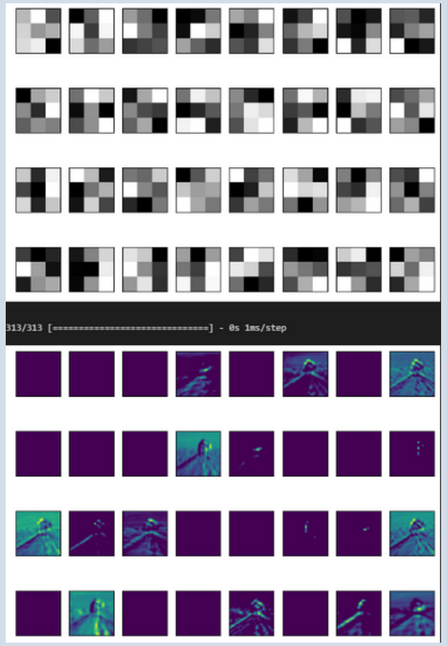
```
X_train = X_train/255
X_val = X_val/255
X_test = X_test/255
```

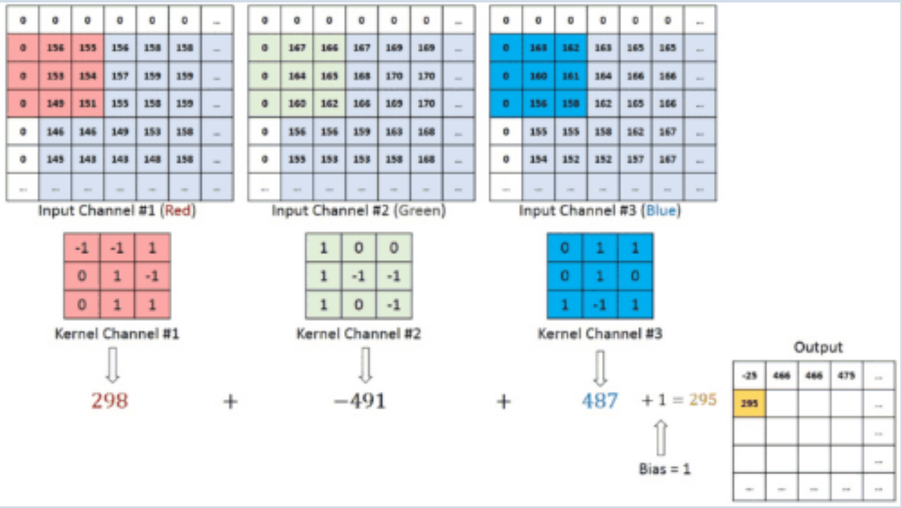
Baseline CNN

Layer (type)	Output Shape	Param #
conv2d_8 (Conv2D)	(None, 30, 30, 32)	896
conv2d_9 (Conv2D)	(None, 28, 28, 64)	18496
flatten_4 (Flatten)	(None, 50176)	0
dense_8 (Dense)	(None, 128)	6422656
dense_9 (Dense)	(None, 100)	12900

Total params: 6,454,948
Trainable params: 6,454,948
Non-trainable params: 0







- 6,454,948 parameters in baseline cnn
- Train accuracy performs well 91.7%
- test accuracy is terribad 11.39%(model is memorising the data it has seen and can't generalise well)
- shows signs of overfitting
- Larger training will help with overfitting

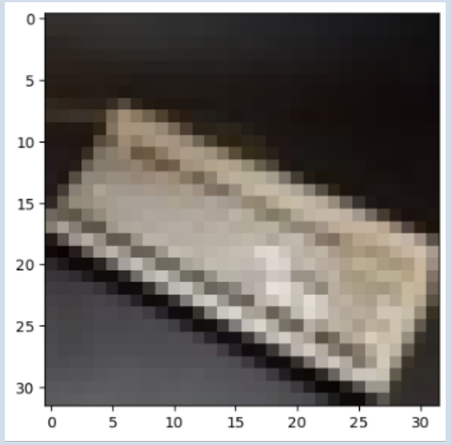
Data augmentation

ImageDataGenerator used

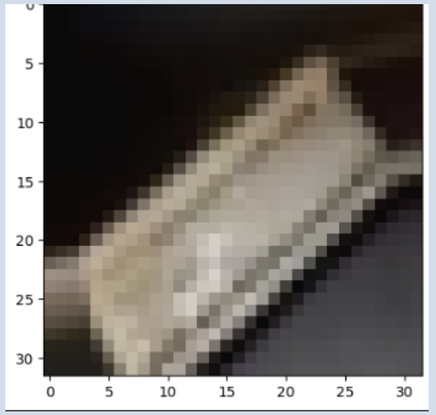
* i did separately for each array

```
datagen = ImageDataGenerator(
    featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    zca_whitening=False,
    rotation_range=90,
    shear_range=0.2,
    zoom_range=0.2,
    channel_shift_range=0.2,
    cval=0.0,
    horizontal_flip=True,
    vertical_flip=True)
```

original



augmented



Data was especially scarce so i delved straight into augmenting first
initially i only augmented 40 000 data points

- feature/sample wise centering/normalizing was not necessary since data was already normalized at the start
- allowed rotation, zooming, shearing,flipping and channel shift (color shifting)
- augmented same number of samples as train set then concatenate with original for training

```
# concat the original and augmented data
X_train = np.concatenate((X_train,X_train_augmented),axis=0)
y_train = np.concatenate((y_train,y_train_augmented),axis=0)

print('after augmentation',X_train.shape,y_train.shape)
```

✓ 0.9s

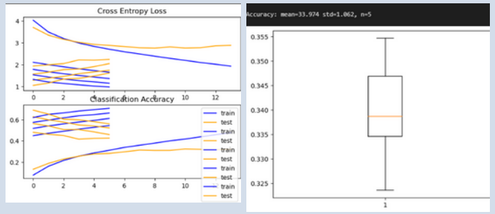
```
after augmentation (400000, 32, 32, 3) (400000, 100)
```

Hypothesis Testing
POOLING

Max Pool

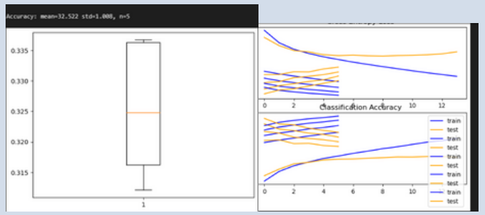
- Max pool worked better for this (takes the brightest for a region of pixels)
- mean 29.652
- std 1.538
- was expecting it be the best

Avg Pool



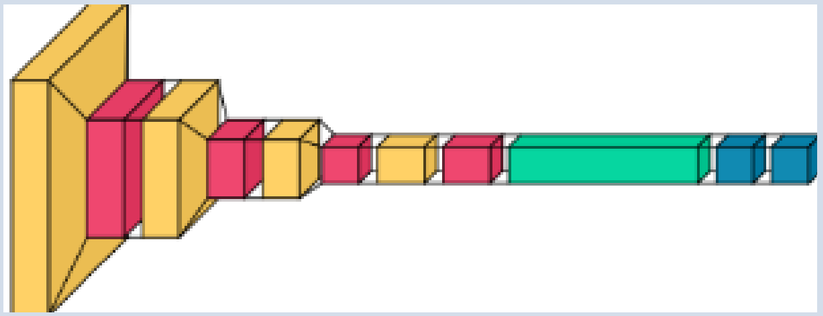
- Average pool takes average of the pixels intensity for a given region
- mean 33.974
- std 1.062
- Overall best alone out of 3
- was surprised it worked better for this dataset

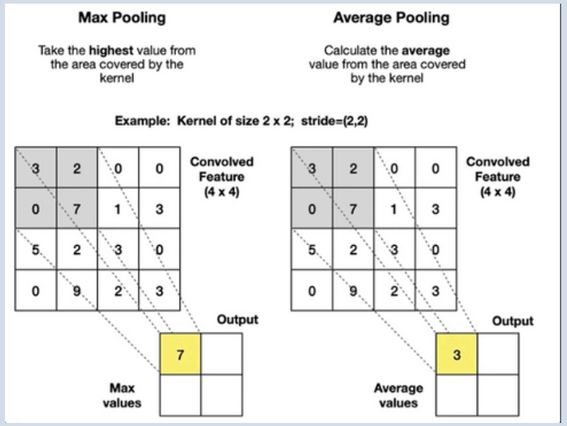
Max + avg Pool



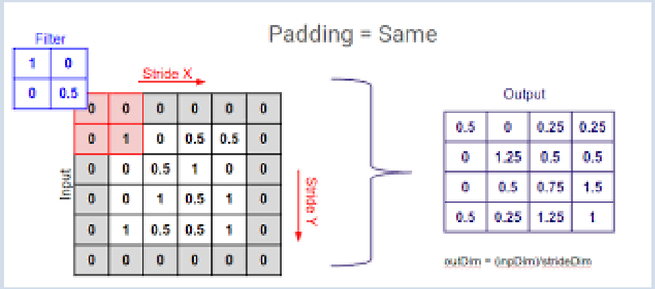
- 2 layers of Max pooling and 2 layers of Average pooling
- Mean 32.522
- std 1.008
- similar in distribution to Average Pool

conv
pool
conv
pool
conv
pool
flatten
Dense
Dense

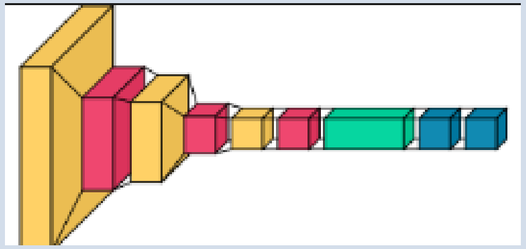




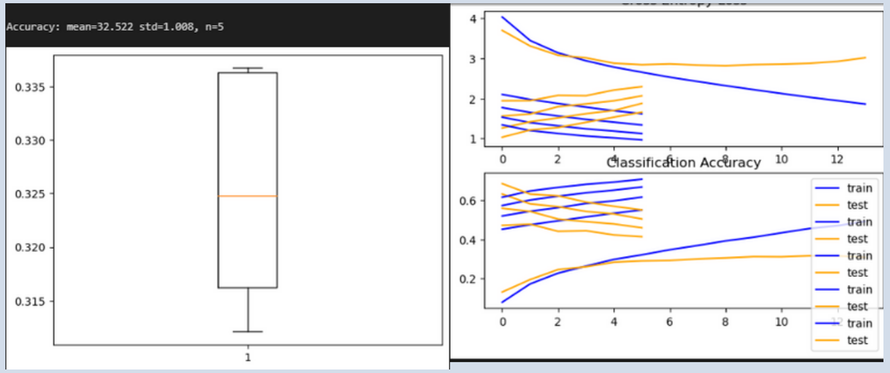
- Reducing spatial size this helps *reduce computational power to process data* (also known as dimensionality reduction)
- Helps in retrieving dominant features
- Done on a simple CNN
- used relu and Adam
- padding was 'same'
- Results may differ after trying out on deeper neural network for cifar 100



PADDING

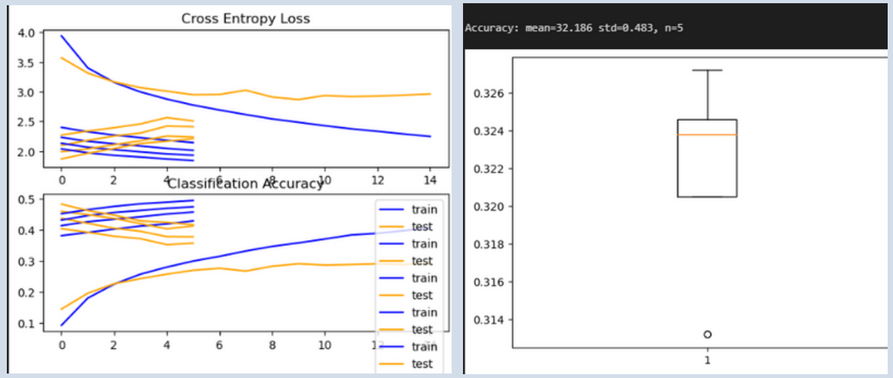


SAME



mean 32.522 std val acc = 1.088

VALID(default)



mean 32.186 , std = 0.483

error

```
ValueError: One of the dimensions in the output is <= 0 due to downsampling in conv2d_3. Consider increasing the input size. Received input shape [None, 32, 32, 3] which would produce output shape with a zero or negative value in a dimension.
```

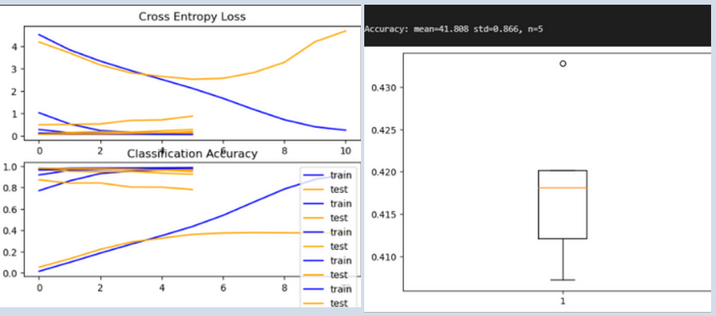
Default is valid padding which means no padding is added to the image.
The image is reduced in size as the filter is moved across the image(depending on stride). However I will have to take note that with valid padding the image will be reduced in size and the model will not be able to learn as much as it would have as compared to models with padding. On top of that , the output shrinks as the filter is moved across the image with a stride of 1 if using valid padding and may cause an error if there are too many convolutional layers because the image will become of size 0 or even negative so will still have to take note of using 'same' padding.

Batch normalization

$$X^* = (X - E[X]) / SQRT(VAR(X))$$

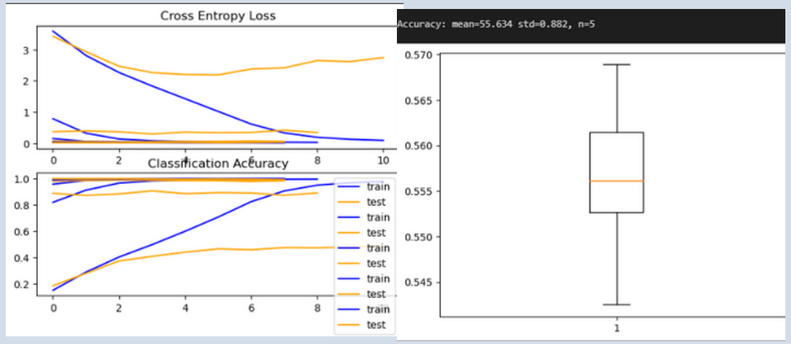
where x * is new value of a single component E[x] is mean in a batch var(x) is variance of x in a batch

without mean 41.8 val acc,std =0.866



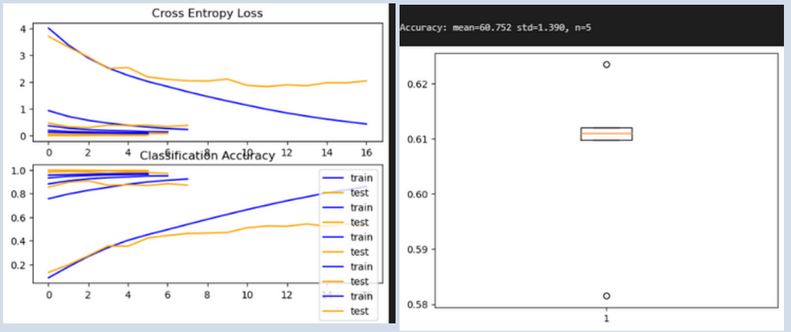
- Model is super overfitted , learning curve started diverging away
- Not being able to generalize well on validation set

with mean = 55.634 val acc ,0.882 std

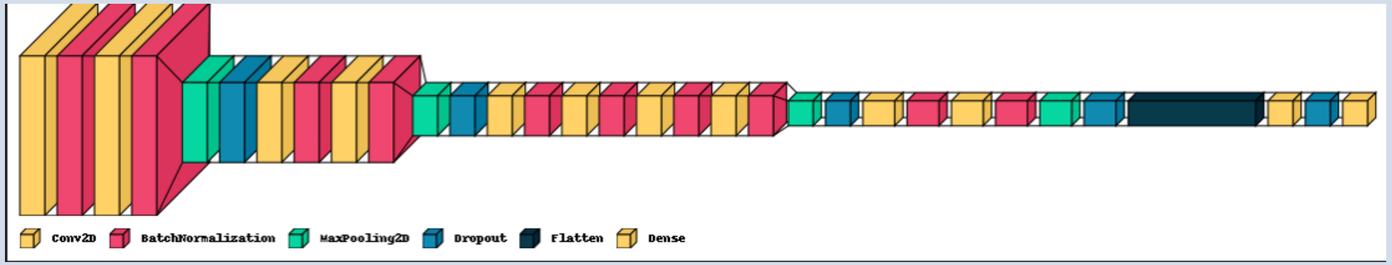


- model begins to improve perhaps due to regularizing effect
- batch norm manages to reduce

dropout + batchnorm
mean = 60.752 val acc, 1.390

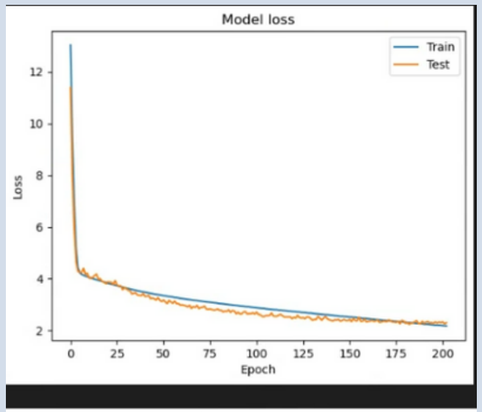


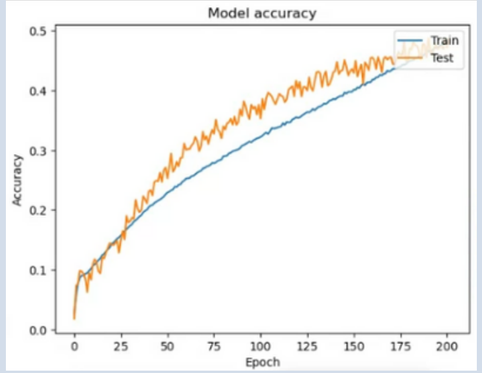
- model performs better with dropout inbetween convolutional layers
- still overfitted
- Adding regularizers next



regularisation

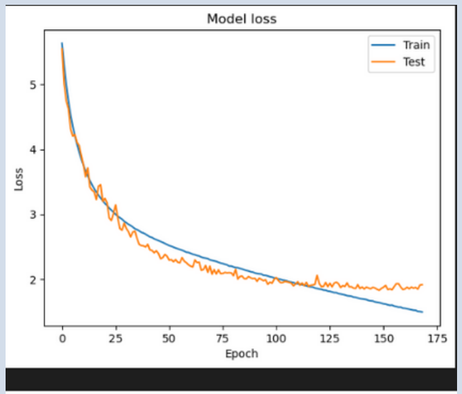
l1

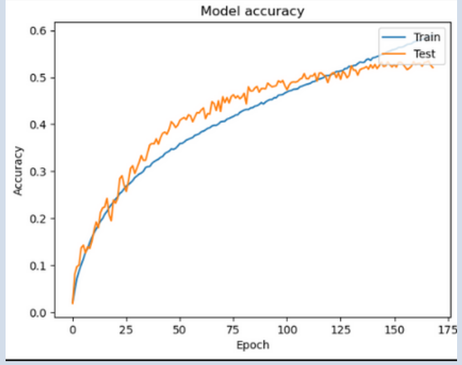




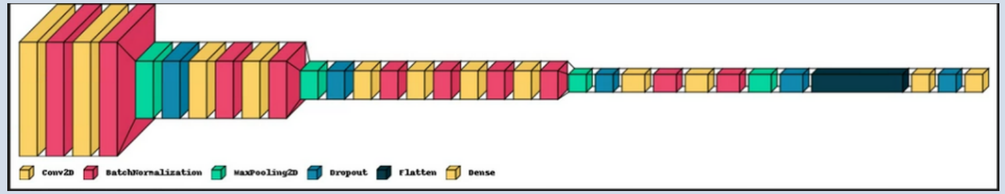
- high epochs but accuracy is not very high
- high loss as well
- ~ 49% val accuracy

L2

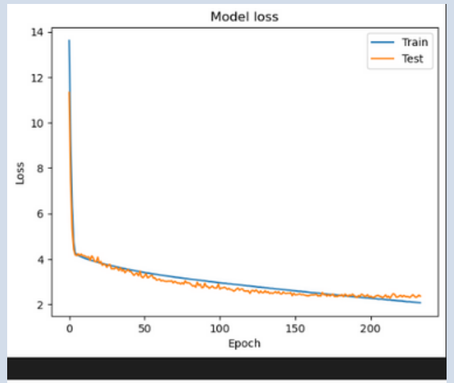


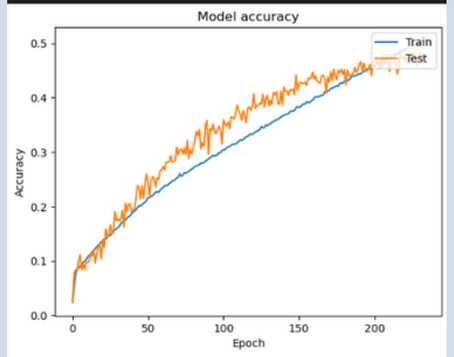


- ~ 55 % val accuracy
- lower loss



L1L2



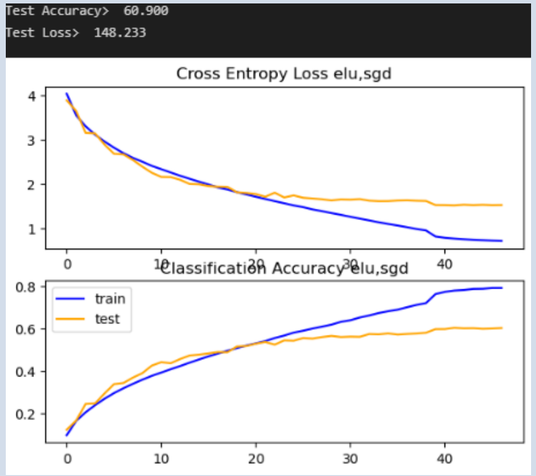


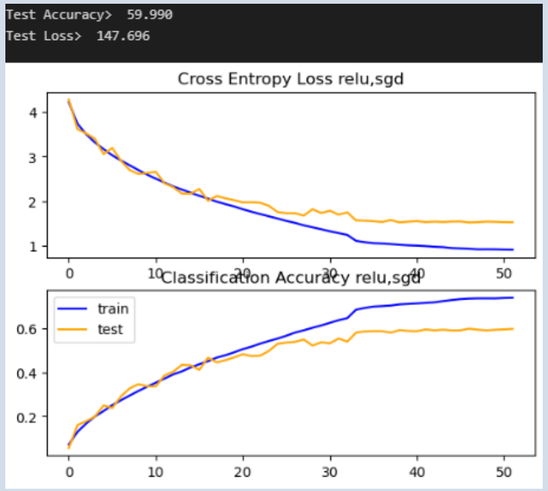
- very similar results to l1
- ~ 49 % val accuracy

tuning activations optimizers

SGD: Performs gradient descent by picking 1 obversation out but also can do in batches and update weights. Has momentum and decay rate
adagrad : adaptive gradient , decreasing learning rate for frequent updates to parameter and bigger steps for rare occurance (has vanishing gradient problem)
adadelta: similar to adagrad but has a window history of past gradients to adapt its learning rate instead of looking at the whole history.
RMSprop: similar to adadelta but uses root mean square to normalize and balance momentum avoid exploding or vanishing gradient
adam(adaptive movement): mix of Adagrad(updates learning rate) and RMSprop for scaling/normalizing gradient
adamax: very similar to adam but considers the infinite norm/ max of past gradient

- Coded itertools function to tune best activation and optimizer pair
- highlighted means best performing pair
- weight initializations matter also affects accuracy so different run gets different results
- Super computationally expensive
- time consuming took 3 days
- early stopping was used

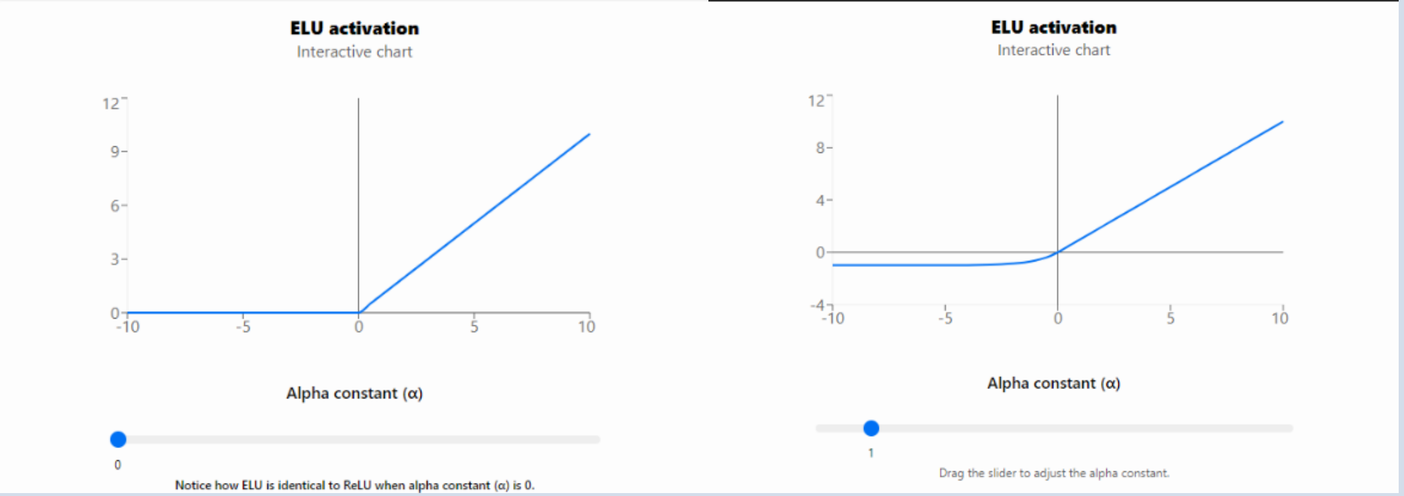




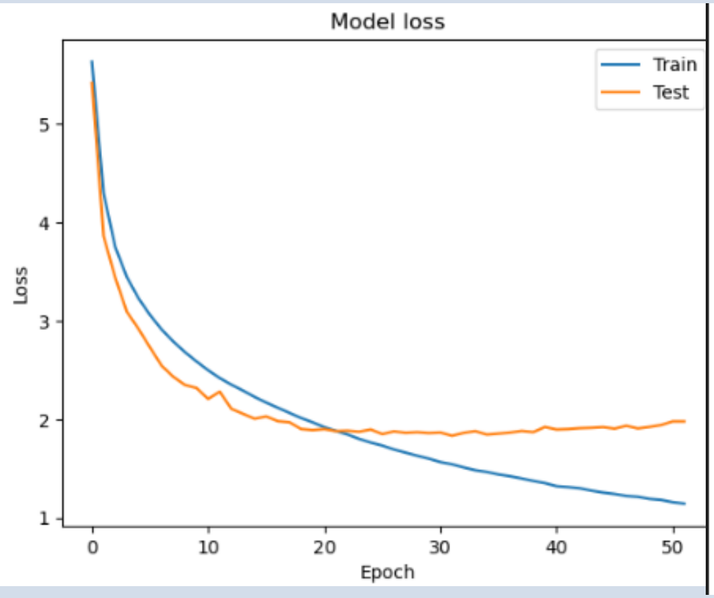
	act_opt	accuracy	loss
0	('relu', 'adam')	36.989999	312.309170
1	('relu', 'sgd')	29.100001	312.991476
2	('relu', 'rmsprop')	38.119999	296.098280
3	('relu', 'adagrad')	26.429999	303.414869
4	('relu', 'adadelta')	13.220000	378.339148
5	('relu', 'adamax')	37.059999	322.905302
6	('relu', 'nadam')	38.139999	320.079541
7	('selu', 'adam')	39.730000	275.166893
8	('selu', 'sgd')	28.580001	296.087503
9	('selu', 'rmsprop')	34.430000	312.057114
10	('selu', 'adagrad')	25.450000	310.959387
11	('selu', 'adadelta')	17.659999	354.994726
12	('selu', 'adamax')	38.690001	287.699580
13	('selu', 'nadam')	38.100001	302.238274
14	('elu', 'adam')	38.929999	275.177550
15	('elu', 'sgd')	37.979999	270.716977
16	('elu', 'rmsprop')	36.730000	309.104967
17	('elu', 'adagrad')	21.330000	333.647656
18	('elu', 'adadelta')	16.280000	359.961295
19	('elu', 'adamax')	38.280001	296.390843
20	('elu', 'nadam')	39.350000	273.205185
21	('tanh', 'adam')	30.919999	285.609913
22	('tanh', 'sgd')	26.580000	299.310184
23	('tanh', 'rmsprop')	24.310000	318.454742
24	('tanh', 'adagrad')	21.940000	328.349137
25	('tanh', 'adadelta')	12.750000	385.026526
26	('tanh', 'adamax')	28.459999	298.511291
27	('tanh', 'nadam')	28.770000	291.105270
28	('LeakyReLU', 'adam')	37.540001	301.249814

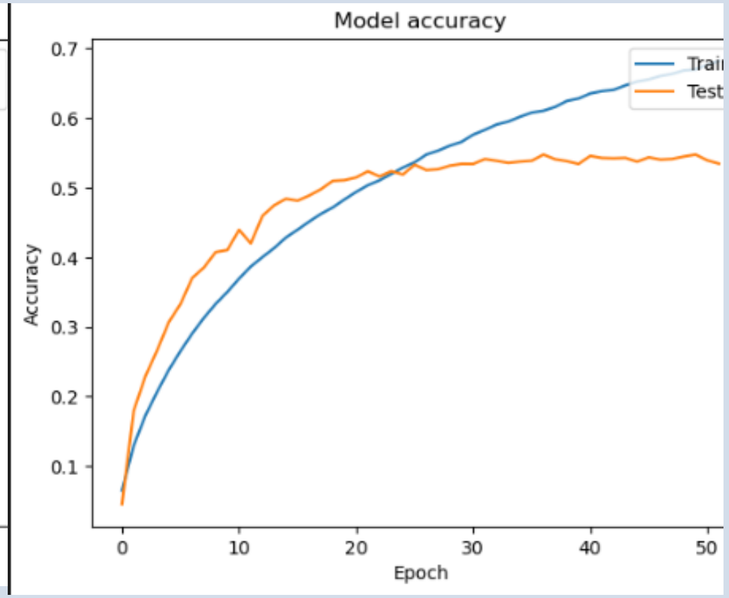
- elu adamax highest val acc
- relu sgd & elu sgd performed decent

Model 1



*elu is similar to relu

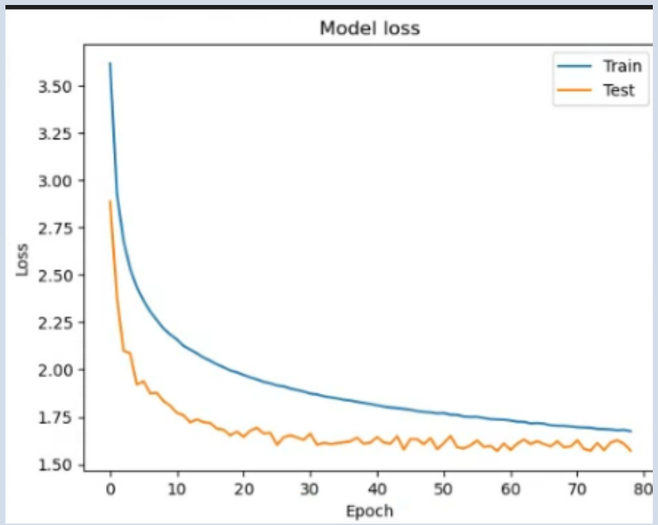


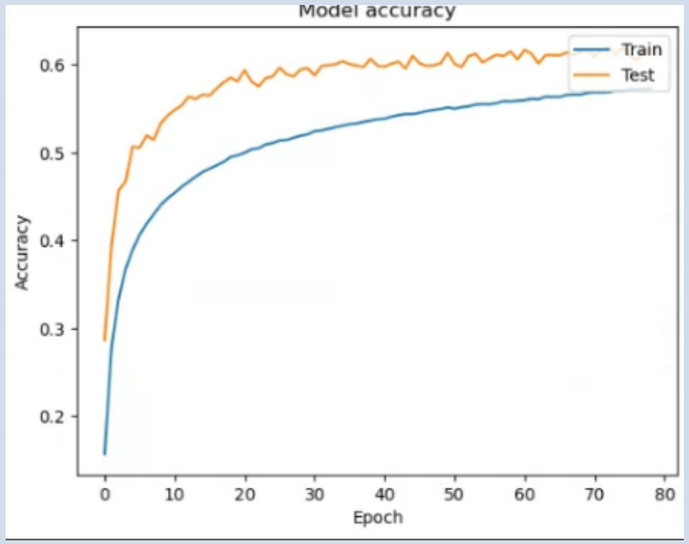


Elu chosen

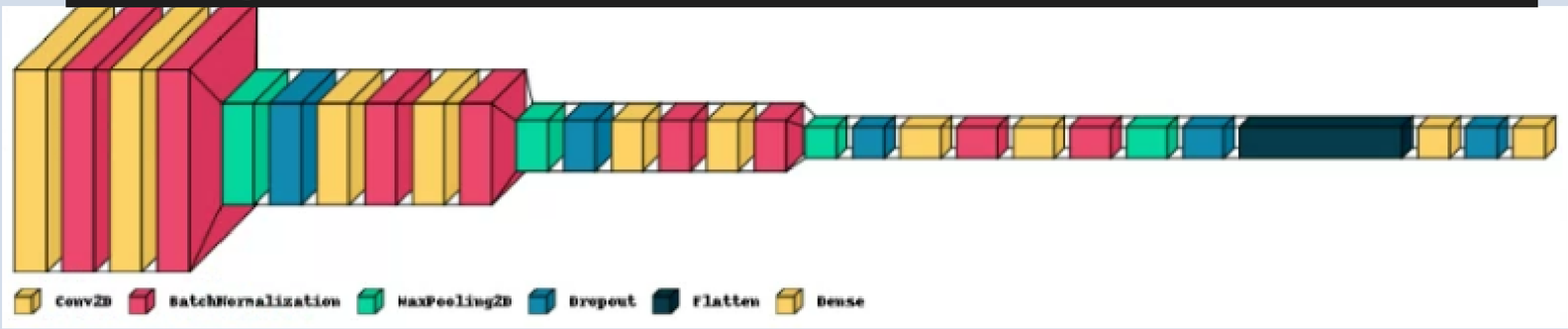
- Faster convergence than relu
- better generalization
- solves vanishing gradients and exploding gradients
- removes dead relu
- 54.76 validation accuracy

final model for fine






```
313/313 [==============================] - 2s 6ms/step - loss: 1.5238 - accuracy: 0.6188
Fine model, accuracy: 61.88%
Fine model, loss: 1.5237715244293213
```



- added more augmented images totaling up to 600 000
- It was the maximum my ram could take
- validation accuracy : 61.98*
- Dropout increased from 0.3 to 0.4
- removed 2 convolutional layer
- reduced l2 from 0.2 to 0.1

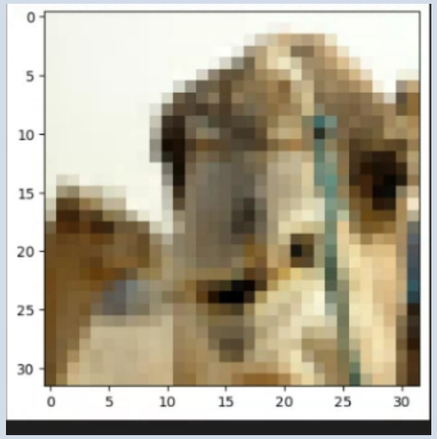
Coarse data

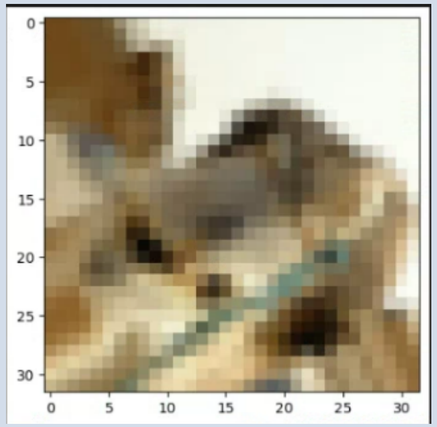
SuperClass
aquatic mammals
fish
flowers
food containers
fruit and vegetables
household electrical devices
household furniture
insects
large carnivores
large man-made outdoor things
large natural outdoor scenes
large omnivores and herbivores
medium-sized mammals
non-insect invertebrates
people
reptiles
small mammals
trees
vehicles_1
vehicles_2

- Images are the same as fine_labels
- coarse contains the superclass version

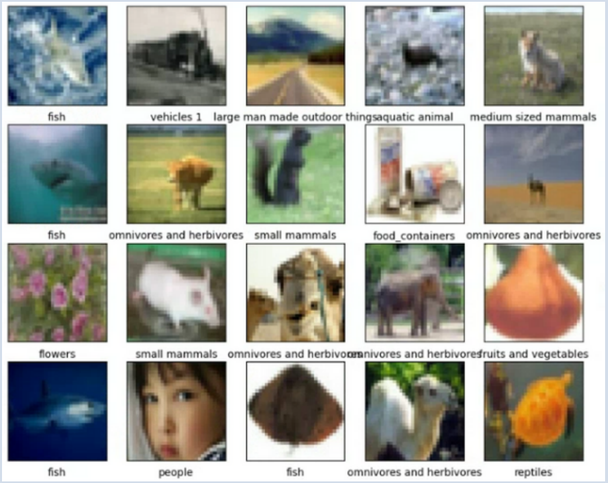
```
y_test_labels = y_test
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_val = to_categorical(y_val)
```

```
X_train = X_train/255
X_val = X_val/255
X_test = X_test/255
```





augmentation of data

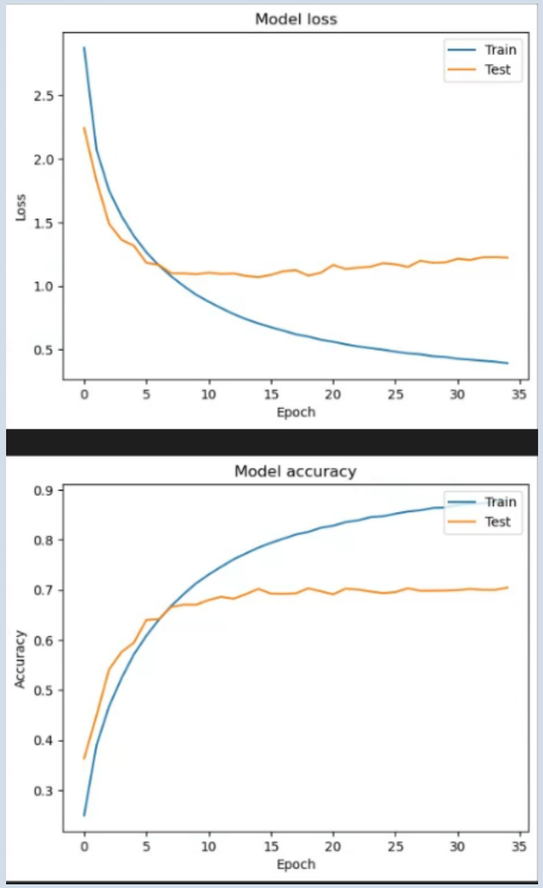


Copied and augment each feature

- 400 000 data points total
- rotating
- shifting
- shearing
- channel shift

Transfer architecture

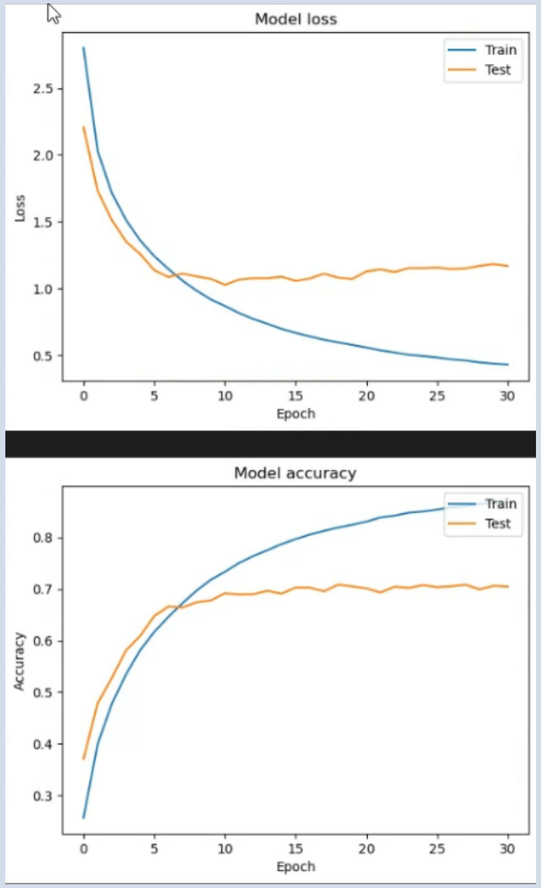
model 1

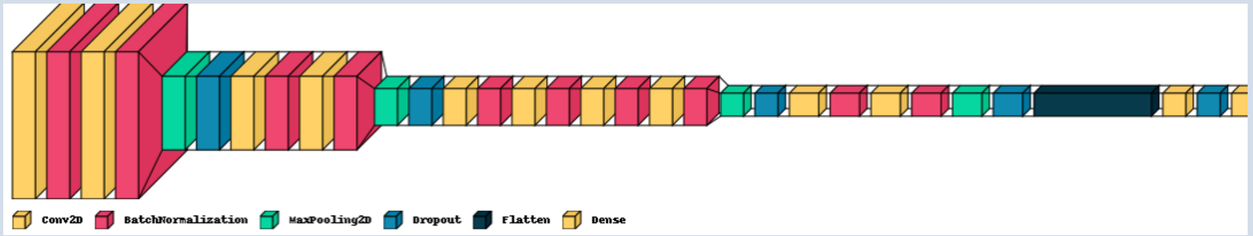


- val accuracy: 70.6%
- 1.1674 val loss
- diverging loss function
- overfitting suggested

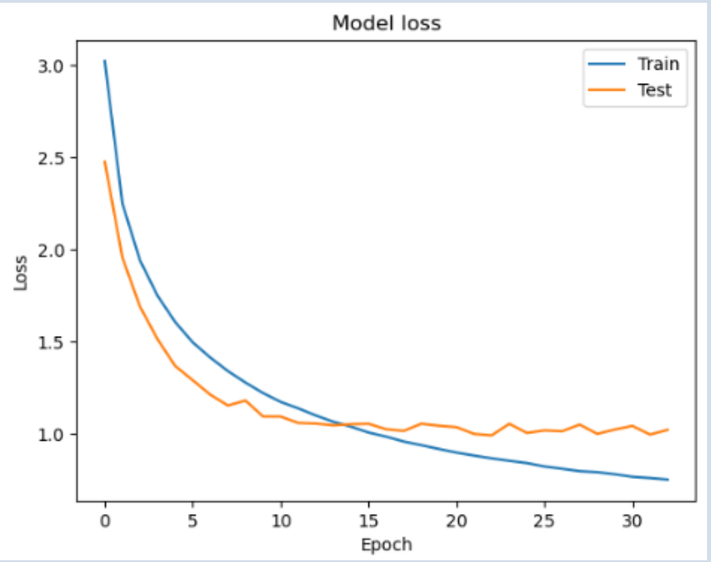
- val accuracy : 70.47%
- 1.1674 val loss
- similar results
- overkill with the layers?
- still overfitted :(

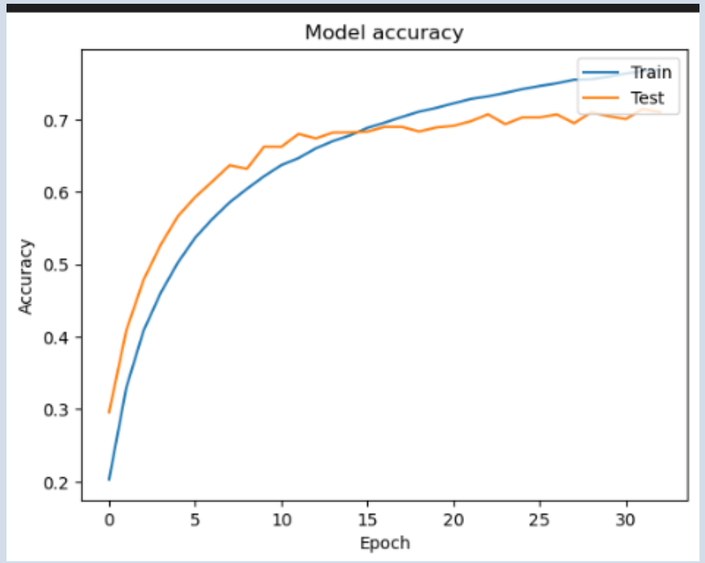
tuned model 2



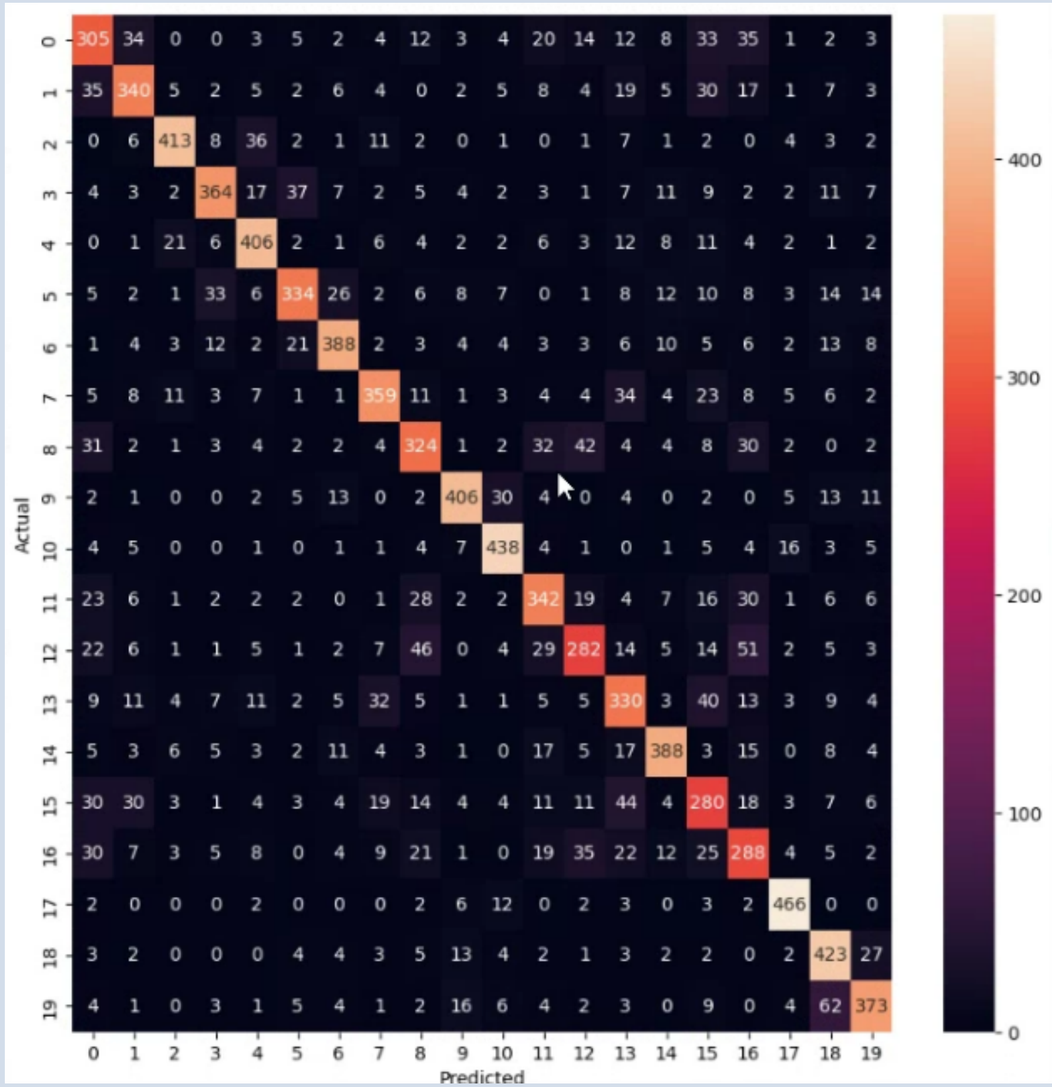


Final Model





Confusion Matrix



- validation accuracy :
- validation loss:
- removed 2 more layers
- dropout is 0.4 for all dropout layers
- L2 = 0.2
- 15 mins to train

*evaluation score:

```
Coarse model, accuracy: 72.49%
Coarse model, loss: 0.9535974860191345
313/313 [==============================] - 2s 5ms/step
```

Conclusion

Explored

hypothesis testing

1. Padding
2. Pooling
3. Batch normalization
4. Dropout
5. regularisation

Tuned

- Activations
- Optimizers
- Dropout rate
- L2 regularizer penalty

Limitations

1. ram space
2. computational power
3. Time to tune every single parameter

Improvements

- More advance architectures like vggnet,efficientnet and alexnet
- More Data (augmentation on the fly but it can take up more time)