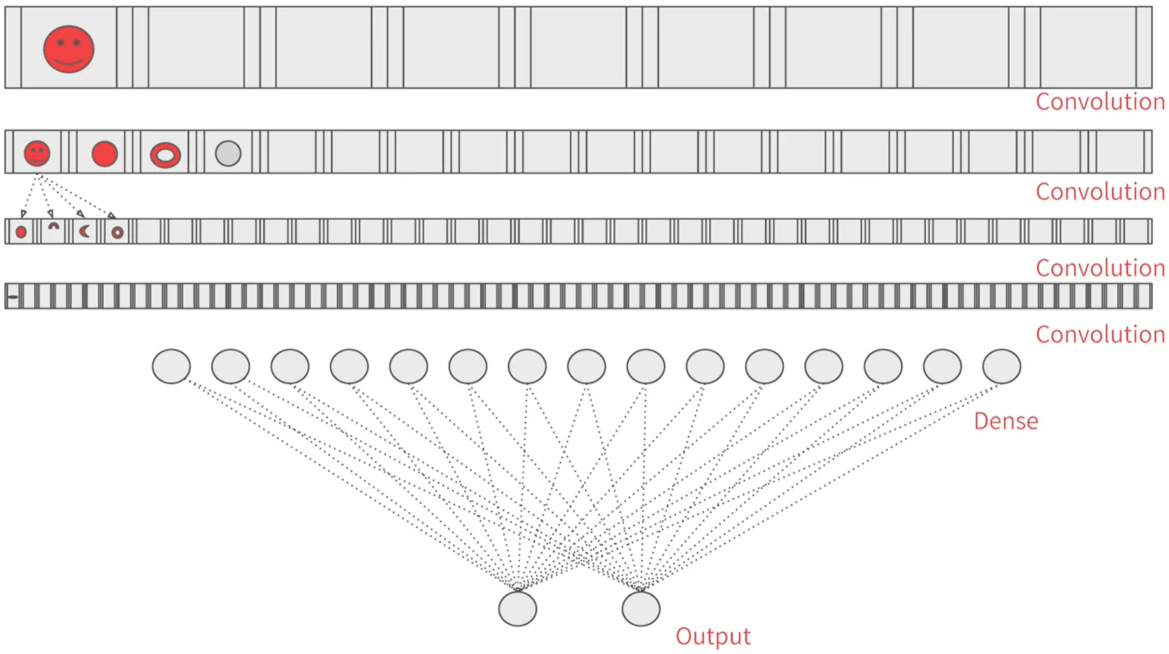
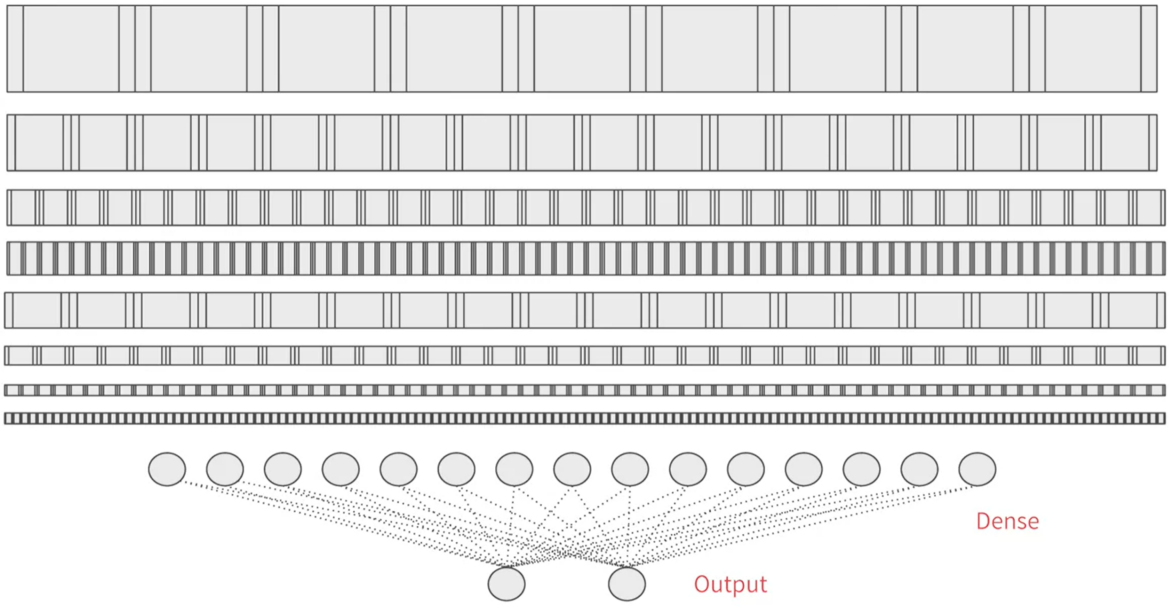
Week 3

Transfer Learning Concept

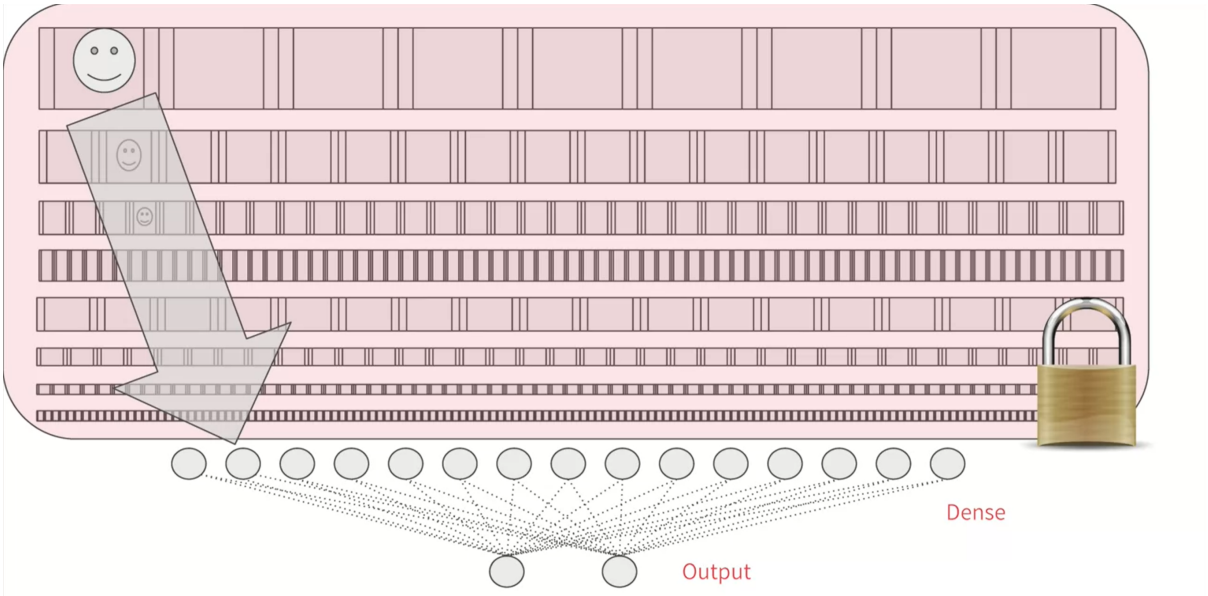
- Even with all the features the model picked up previously within the 1500 images, it will not be enough.
- Transfer learning takes features from models people trained and appying it to your own model
- The image below might be the network you have created



- But consider someone else's model that is much more sophisticated than yours and trained a lot more layers



- Then you can take the features the network learned and apply it into your own network



- This way you would have a model that is trained on a very large dataset and use the convolution it learn while it's classifying
- You don't need to use all the layers since some layers down bottom may be very specific to the model

Code Implementation

- Imports needed for the code

```
import os

from tensorflow.keras import layers
from tensorflow.keras import Model
```

- The pretrained weights are in the link below
- This is a snapshot of model after being trained

```
https://storage.googleapis.com/mledu-datasets/
 inception_v3_weights_tf_dim_ordering_tf_kernels
```

- specify not using the built in weight but using the inception weight

```
from tensorflow.keras.applications.inception_v3 import InceptionV3

local_weights_file = '/tmp/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5'

pre_trained_model = InceptionV3(input_shape = (150, 150, 3),
                                include_top = False,
                                weights = None)

pre_trained_model.load_weights(local_weights_file)
```

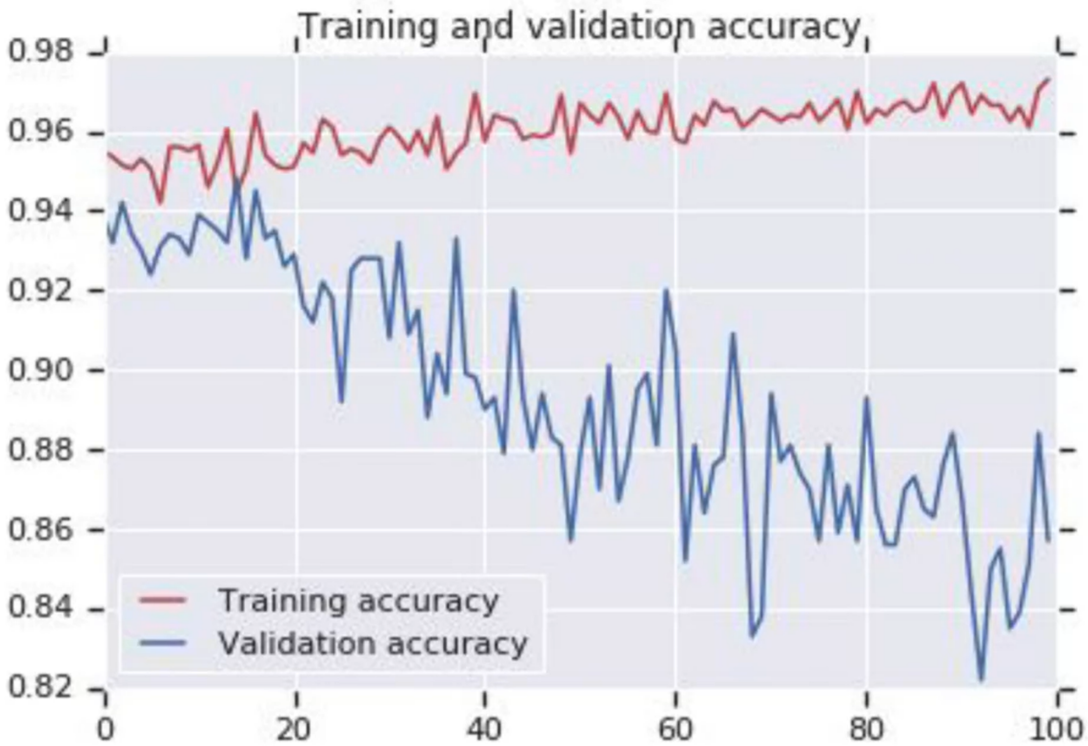
- All of the layers have names

```
last_layer = pre_trained_model.get_layer('mixed7')

last_output = last_layer.output
```

- Using this layer

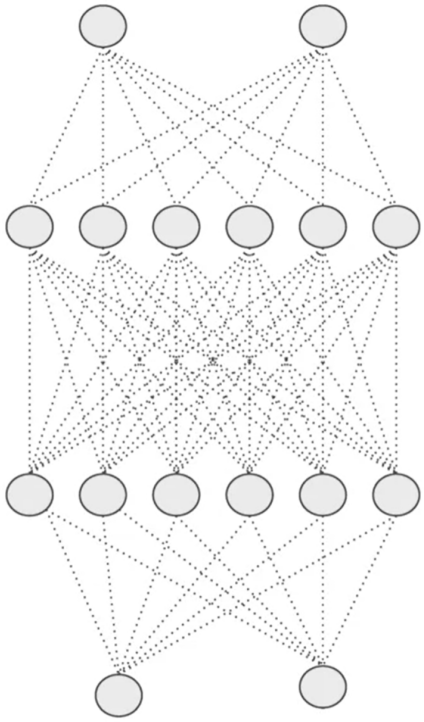
```
from tensorflow.keras.optimizers import RMSprop

x = layers.Flatten()(last_output)
x = layers.Dense(1024, activation='relu')(x)
x = layers.Dense  (1, activation='sigmoid')(x)

model = Model( pre_trained_model.input, x)
model.compile(optimizer = RMSprop(lr=0.0001),
              loss = 'binary_crossentropy',
              metrics = ['acc'])
```

- After training and validating the data with 100 epochs
- The model is overfitting

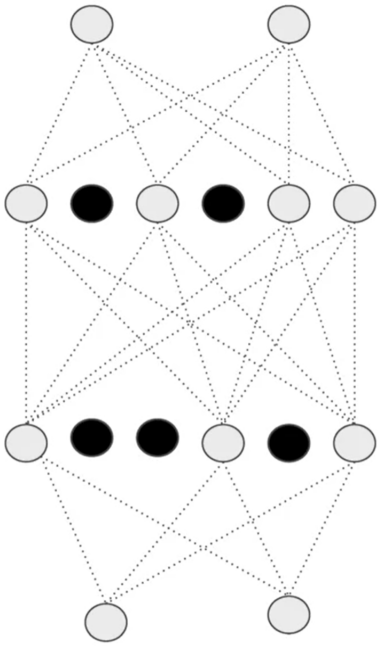


- A solution to the overfitting issue could be dropouts

- The idea of dropouts is to remove neurons that's similar that could affect each other in a bad way



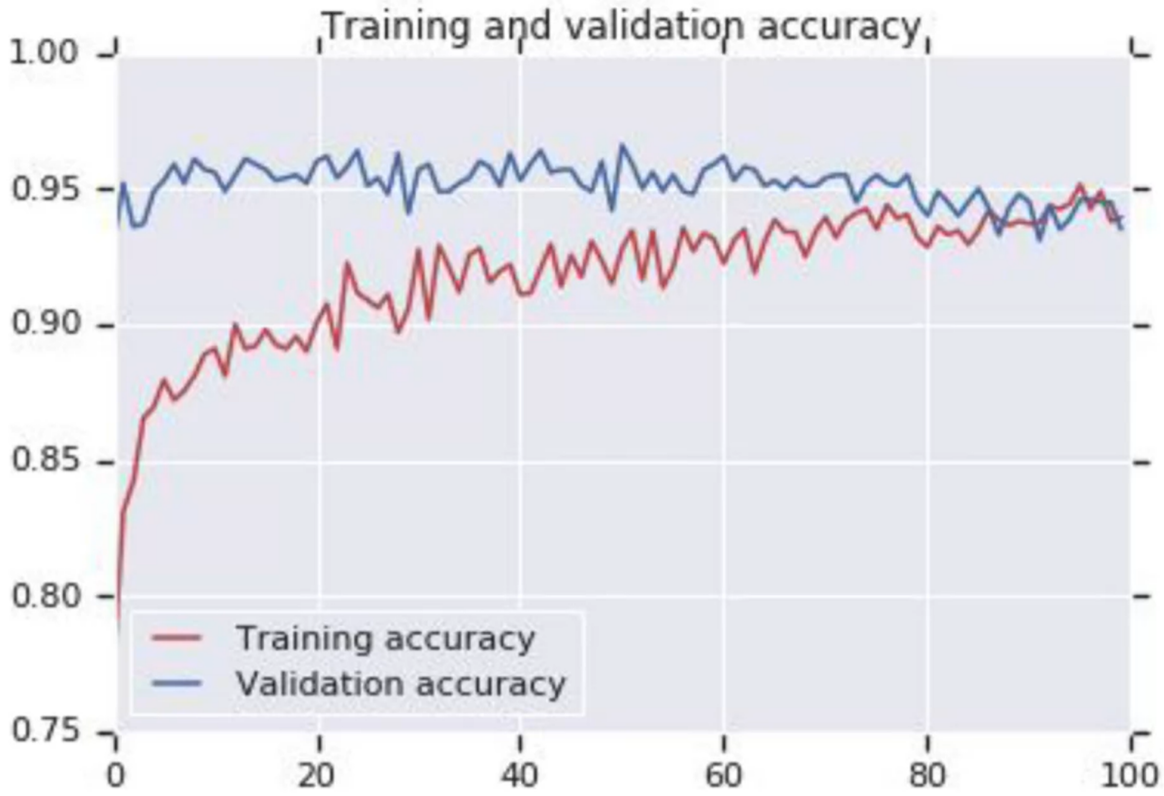
- The diagram below shows the network after dropout



- The code below is how we implement this in code

```
from tensorflow.keras.optimizers import RMSprop

x = layers.Flatten()(last_output)
x = layers.Dense(1024, activation='relu')(x)
x = layers.Dropout(0.2)(x)
x = layers.Dense  (1, activation='sigmoid')(x)

model = Model( pre_trained_model.input, x)
model.compile(optimizer = RMSprop(lr=0.0001),
              loss = 'binary_crossentropy',
              metrics = ['acc'])
```

- The 0.2 in the dropout function means dropping 20 percent of the neurons
- The graph below shows the accuracy after drop outs



Code For This week

—FILE—

Test

—FILE—