EECS 738 Lab 4

Step 0: Import relevant packages

In [52]:
```
from __future__ import print_function

import os

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import *
from tensorflow.keras.optimizers import SGD, Adam
from tensorflow.keras.activations import relu
from tensorflow.keras.regularizers import l2
from tensorflow.keras.constraints import max_norm
from tensorflow.keras import backend as K
from tensorflow.keras.datasets import mnist, cifar10
from keras.callbacks import ModelCheckpoint, LearningRateScheduler
from keras.callbacks import ReduceLROnPlateau

print('The Tensorflow version is {}.'.format(tf.__version__))
print('The Keras version is {}.'.format(keras.__version__))
print('The Pandas version is {}.'.format(pd.__version__))

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

print("Packages Loaded")

def lr_schedule(epoch):
    """Learning Rate Schedule

    Learning rate is scheduled to be reduced after 80, 120, 160, 180 epochs.
    Called automatically every epoch as part of callbacks during training.

    # Arguments
        epoch (int): The number of epochs

    # Returns
        lr (float32): learning rate
    """
```

```
    lr = 1e-3
    if epoch > 25:
        lr *= 0.5e-3
    elif epoch > 20:
        lr *= 1e-3
    elif epoch > 15:
        lr *= 1e-2
    elif epoch > 10:
        lr *= 1e-1
    print('Learning rate: ', lr)
    return lr
```

```
The Tensorflow version is 2.1.0.
The Keras version is 2.2.4-tf.
The Pandas version is 0.23.4.
Packages Loaded
```

Step 1: Load data in to train and test splits

In [53]:

```
# Lets import our MNIST data like we did in lab 3.
mnist = tf.keras.datasets.mnist
cifar = tf.keras.datasets.cifar10
(mx_train, my_train), (mx_test, my_test) = mnist.load_data()
(cx_train, cy_train), (cx_test, cy_test) = cifar.load_data()

print('Shape of mnist train data {}.'.format(mx_train.shape))
print('Shape of mnist test data {}.'.format(mx_test.shape))

print('Shape of cifar train data {}.'.format(cx_train.shape))
print('Shape of cifar test data {}.'.format(cx_test.shape))

print(type(mx_test))
```

```
Shape of mnist train data (60000, 28, 28).
Shape of mnist test data (10000, 28, 28).
Shape of cifar train data (50000, 32, 32, 3).
Shape of cifar test data (10000, 32, 32, 3).
<class 'numpy.ndarray'>
```

Step2: Prepare the data

In [54]:
```
#Normalize the data
mx_train = mx_train/255
mx_test = mx_test / 255

mx_train = tf.keras.utils.normalize(mx_train, axis=1)
mx_test = tf.keras.utils.normalize(mx_test, axis=1)

mx_train_hold = mx_train
my_train_hold = my_train

cx_train = tf.keras.utils.normalize(cx_train, axis=1)
cx_test = tf.keras.utils.normalize(cx_test, axis=1)

cx_train = cx_train.astype('float32')
cx_test = cx_test.astype('float32')
cx_train /= 255
cx_test /= 255

#im = mx_train[4,:,:]
#plt.imshow(im, cmap=plt.cm.binary)

#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')


#y_train = tf.keras.utils.to_categorical(y_train, 10)
#y_test = tf.keras.utils.to_categorical(y_test, 10)
# Now import Cifar-10 data and process it.
#(train, target), (test, test_target) = cifar10.load_data()
# Fill in the rest.
```

Step 3: Sequential NN shape input, 64, 64, 64, 10

In [4]:
```
# Our baseline model for this lab.
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28,28]))

model.add(keras.layers.Dense(64, activation="relu"))
model.add(keras.layers.Dense(64, activation="relu"))
model.add(keras.layers.Dense(64, activation="relu"))
model.add(keras.layers.Dense(10, activation="softmax"))


model.compile(loss='sparse_categorical_crossentropy',
              optimizer=SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True),
              metrics=['accuracy'])

#model.fit(mx_train, my_train, epochs=20, batch_size=64, validation_data=(mx_test, my_test))
model_detail = model.fit(mx_train, my_train, epochs=20, batch_size=64, validation_split=0.1)

#plot accuracies for each epoch
history = pd.DataFrame(model_detail.history)

history.plot(figsize=(10,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.xlabel('Epochs')
plt.title('Sequential Neural Nets')
plt.show()

#After training the model, evaluate the test set
model.evaluate(mx_test,my_test)

#Print the summary of the model
model.summary()
```

```
Train on 54000 samples, validate on 6000 samples
Epoch 1/20
54000/54000 [==============================] - 2s 32us/sample - loss: 0.5926 - accuracy: 0.8233 - val_loss:
0.2111 - val_accuracy: 0.9410
Epoch 2/20
54000/54000 [==============================] - 1s 27us/sample - loss: 0.2171 - accuracy: 0.9357 - val_loss:
0.1543 - val_accuracy: 0.9538
Epoch 3/20
54000/54000 [==============================] - 2s 28us/sample - loss: 0.1579 - accuracy: 0.9523 - val_loss:
0.1278 - val_accuracy: 0.9632
Epoch 4/20
54000/54000 [==============================] - 1s 27us/sample - loss: 0.1261 - accuracy: 0.9624 - val_loss:
0.1137 - val_accuracy: 0.9665
Epoch 5/20
54000/54000 [==============================] - 1s 28us/sample - loss: 0.1049 - accuracy: 0.9691 - val_loss:
0.1141 - val_accuracy: 0.9673
Epoch 6/20
54000/54000 [==============================] - 1s 27us/sample - loss: 0.0888 - accuracy: 0.9729 - val_loss:
0.1021 - val_accuracy: 0.9700
Epoch 7/20
54000/54000 [==============================] - 1s 27us/sample - loss: 0.0762 - accuracy: 0.9765 - val_loss:
0.0990 - val_accuracy: 0.9710
Epoch 8/20
54000/54000 [==============================] - 1s 27us/sample - loss: 0.0661 - accuracy: 0.9797 - val_loss:
0.0988 - val_accuracy: 0.9710
Epoch 9/20
54000/54000 [==============================] - 2s 28us/sample - loss: 0.0564 - accuracy: 0.9825 - val_loss:
0.0935 - val_accuracy: 0.9727
Epoch 10/20
54000/54000 [==============================] - 2s 29us/sample - loss: 0.0514 - accuracy: 0.9841 - val_loss:
0.0952 - val_accuracy: 0.9722
Epoch 11/20
54000/54000 [==============================] - 1s 28us/sample - loss: 0.0452 - accuracy: 0.9866 - val_loss:
0.0951 - val_accuracy: 0.9730
Epoch 12/20
54000/54000 [==============================] - 2s 28us/sample - loss: 0.0400 - accuracy: 0.9867 - val_loss:
0.1031 - val_accuracy: 0.9703
Epoch 13/20
54000/54000 [==============================] - 2s 28us/sample - loss: 0.0347 - accuracy: 0.9894 - val_loss:
0.1052 - val_accuracy: 0.9717
Epoch 14/20
54000/54000 [==============================] - 1s 27us/sample - loss: 0.0302 - accuracy: 0.9907 - val_loss:
0.1083 - val_accuracy: 0.9698
```

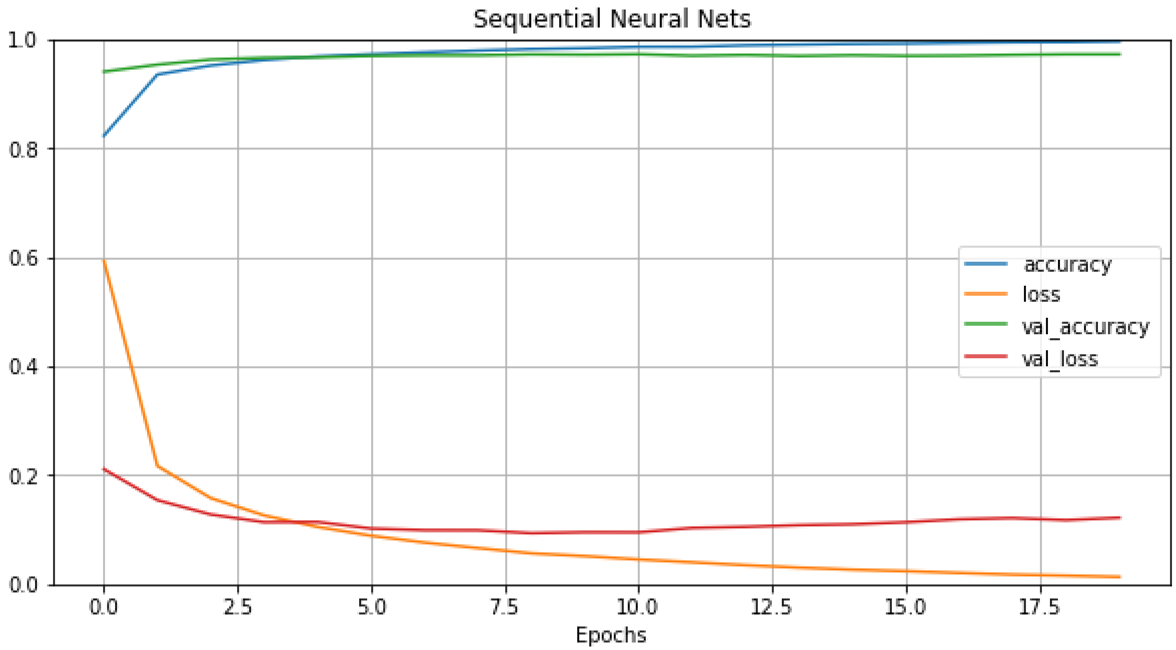
```
Epoch 15/20
54000/54000 [==============================] - 1s 27us/sample - loss: 0.0265 - accuracy: 0.9920 - val_loss:
0.1099 - val_accuracy: 0.9712
Epoch 16/20
54000/54000 [==============================] - 2s 28us/sample - loss: 0.0237 - accuracy: 0.9926 - val_loss:
0.1136 - val_accuracy: 0.9700
Epoch 17/20
54000/54000 [==============================] - 1s 27us/sample - loss: 0.0205 - accuracy: 0.9939 - val_loss:
0.1191 - val_accuracy: 0.9705
Epoch 18/20
54000/54000 [==============================] - 1s 27us/sample - loss: 0.0174 - accuracy: 0.9950 - val_loss:
0.1213 - val_accuracy: 0.9718
Epoch 19/20
54000/54000 [==============================] - 1s 27us/sample - loss: 0.0155 - accuracy: 0.9954 - val_loss:
0.1175 - val_accuracy: 0.9728
Epoch 20/20
54000/54000 [==============================] - 2s 30us/sample - loss: 0.0131 - accuracy: 0.9966 - val_loss:
0.1219 - val_accuracy: 0.9728
```

```
Out[4]: <matplotlib.axes._subplots.AxesSubplot at 0x26dbce3feb8>

Out[4]: (0, 1)

Out[4]: Text(0.5, 0, 'Epochs')

Out[4]: Text(0.5, 1.0, 'Sequential Neural Nets')
```




```
10000/10000 [==============================] - 0s 32us/sample - loss: 0.1254 - accuracy: 0.9689
```

Out[4]:
```
[0.12535318766142883, 0.9689]
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 64)                50240
_________________________________________________________________
dense_1 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_3 (Dense)              (None, 10)                650
=================================================================
Total params: 59,210
Trainable params: 59,210
Non-trainable params: 0
_________________________________________________________________
```

Step 4: Convert this baseline into a Functional Model using keras' Functional Model API.

[https://keras.io/getting-started/functional-api-guide/ (https://keras.io/getting-started/functional-api-guide/)](https://keras.io/getting-started/functional-api-guide/)

In [9]:
```
# Create the functional Baseline here.
input_layer = keras.layers.Input(shape=mx_train.shape[1:])
il = Flatten()(input_layer)
h1 = keras.layers.Dense(64,activation="relu")(il)
h2 = keras.layers.Dense(64,activation="relu")(h1)
h3 = keras.layers.Dense(64,activation="relu")(h2)
output_layer = keras.layers.Dense(10, activation="softmax")(h3)
fmodel = keras.models.Model(inputs=[input_layer], outputs=[output_layer])

fmodel.compile(loss='sparse_categorical_crossentropy',
              optimizer=keras.optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True),metrics=['accu
racy'])

fmodel_detail = fmodel.fit(mx_train, my_train, epochs=20, batch_size=64, validation_split=0.1)

#plot accuracies for each epoch
fhistory = pd.DataFrame(fmodel_detail.history)
fhistory.plot(figsize=(10,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.xlabel('Epochs')
plt.title('Functional Neural Nets')
plt.show()

#After training the model, evaluate the test set
fmodel.evaluate(mx_test,my_test)

#Print the summary of the model
fmodel.summary()
```

```
Train on 54000 samples, validate on 6000 samples
Epoch 1/20
54000/54000 [==============================] - 2s 31us/sample - loss: 0.6214 - accuracy: 0.8124 - val_loss:
0.2098 - val_accuracy: 0.9402
Epoch 2/20
54000/54000 [==============================] - 2s 28us/sample - loss: 0.2175 - accuracy: 0.9365 - val_loss:
0.1518 - val_accuracy: 0.9555
Epoch 3/20
54000/54000 [==============================] - 1s 28us/sample - loss: 0.1595 - accuracy: 0.9524 - val_loss:
0.1198 - val_accuracy: 0.9625
Epoch 4/20
54000/54000 [==============================] - 1s 26us/sample - loss: 0.1259 - accuracy: 0.9619 - val_loss:
0.1091 - val_accuracy: 0.9670
Epoch 5/20
54000/54000 [==============================] - 2s 30us/sample - loss: 0.1026 - accuracy: 0.9693 - val_loss:
0.1113 - val_accuracy: 0.9672
Epoch 6/20
54000/54000 [==============================] - 2s 29us/sample - loss: 0.0883 - accuracy: 0.9734 - val_loss:
0.1065 - val_accuracy: 0.9700
Epoch 7/20
54000/54000 [==============================] - 1s 26us/sample - loss: 0.0761 - accuracy: 0.9764 - val_loss:
0.0951 - val_accuracy: 0.9712
Epoch 8/20
54000/54000 [==============================] - 1s 28us/sample - loss: 0.0651 - accuracy: 0.9801 - val_loss:
0.1056 - val_accuracy: 0.9680
Epoch 9/20
54000/54000 [==============================] - 1s 28us/sample - loss: 0.0569 - accuracy: 0.9823 - val_loss:
0.0982 - val_accuracy: 0.9722
Epoch 10/20
54000/54000 [==============================] - 1s 27us/sample - loss: 0.0497 - accuracy: 0.9850 - val_loss:
0.0925 - val_accuracy: 0.9742
Epoch 11/20
54000/54000 [==============================] - 1s 27us/sample - loss: 0.0440 - accuracy: 0.9863 - val_loss:
0.0969 - val_accuracy: 0.9740
Epoch 12/20
54000/54000 [==============================] - 2s 30us/sample - loss: 0.0389 - accuracy: 0.9879 - val_loss:
0.0936 - val_accuracy: 0.9757
Epoch 13/20
54000/54000 [==============================] - 1s 28us/sample - loss: 0.0342 - accuracy: 0.9896 - val_loss:
0.0960 - val_accuracy: 0.9738
Epoch 14/20
54000/54000 [==============================] - 1s 27us/sample - loss: 0.0298 - accuracy: 0.9909 - val_loss:
0.1034 - val_accuracy: 0.9745
```

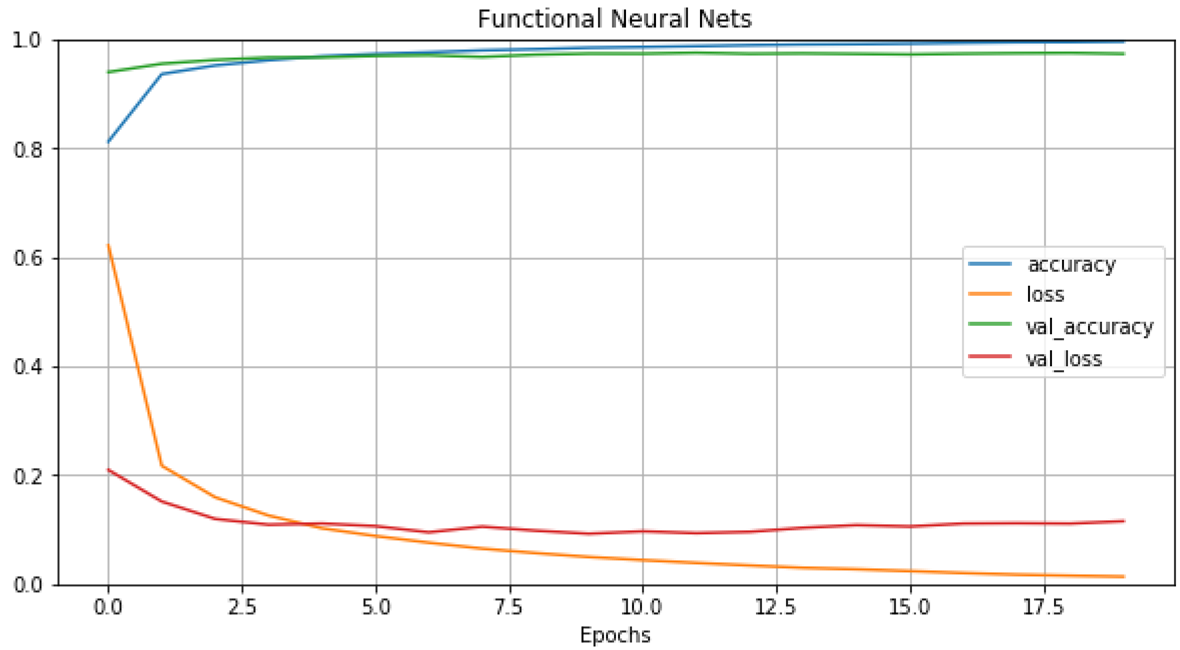
```
Epoch 15/20
54000/54000 [==============================] - 1s 28us/sample - loss: 0.0272 - accuracy: 0.9915 - val_loss:
0.1083 - val_accuracy: 0.9737
Epoch 16/20
54000/54000 [==============================] - 2s 28us/sample - loss: 0.0238 - accuracy: 0.9926 - val_loss:
0.1058 - val_accuracy: 0.9727
Epoch 17/20
54000/54000 [==============================] - 1s 26us/sample - loss: 0.0203 - accuracy: 0.9939 - val_loss:
0.1112 - val_accuracy: 0.9740
Epoch 18/20
54000/54000 [==============================] - 1s 27us/sample - loss: 0.0173 - accuracy: 0.9950 - val_loss:
0.1118 - val_accuracy: 0.9748
Epoch 19/20
54000/54000 [==============================] - 1s 27us/sample - loss: 0.0154 - accuracy: 0.9954 - val_loss:
0.1111 - val_accuracy: 0.9753
Epoch 20/20
54000/54000 [==============================] - 2s 28us/sample - loss: 0.0136 - accuracy: 0.9963 - val_loss:
0.1155 - val_accuracy: 0.9738
```

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x26de10dba58>

Out[9]: (0, 1)

Out[9]: Text(0.5, 0, 'Epochs')

Out[9]: Text(0.5, 1.0, 'Functional Neural Nets')




```
10000/10000 [==============================] - 0s 28us/sample - loss: 0.1253 - accuracy: 0.9690
```

Out[9]:
```
[0.12528867619766387, 0.969]
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_2 (InputLayer)         [(None, 784)]             0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 784)               0         
_________________________________________________________________
dense_43 (Dense)             (None, 64)                50240     
_________________________________________________________________
dense_44 (Dense)             (None, 64)                4160      
_________________________________________________________________
dense_45 (Dense)             (None, 64)                4160      
_________________________________________________________________
dense_46 (Dense)             (None, 10)                650       
=================================================================
Total params: 59,210
Trainable params: 59,210
Non-trainable params: 0
_________________________________________________________________
```

Step 5: Shallow ResNet for MNIST.

In [6]:
```
inputs = tf.keras.Input(shape=(784,), name='img')
x = Dense(128, activation='relu')(inputs)
block_1_output = Dense(128, activation='relu')(x)

x = Dense(128)(block_1_output)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)
x = Dense(128)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)

outputs = Dense(10, activation='softmax')(x)

rmodel = tf.keras.Model(inputs, outputs, name='resnet')

rmodel.compile(Adam(amsgrad=True), 'sparse_categorical_crossentropy', metrics=['accuracy'])

mx_train = mx_train.reshape(60000, 784)
mx_test = mx_test.reshape(10000, 784)
#mx_train = mx_train.astype('float32')
#mx_test = mx_test.astype('float32')

rmodel_detail = rmodel.fit(mx_train, my_train,
              batch_size=128,
              epochs=20,
              validation_split=0.1)

#plot accuracies for each epoch
rhistory = pd.DataFrame(rmodel_detail.history)
rhistory.plot(figsize=(10,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.xlabel('Epochs')
plt.title('Residual Neural Nets')
plt.show()

#After training the model, evaluate the test set
rmodel.evaluate(mx_test,my_test)
```

```
#Print the summary of the model
rmodel.summary()
```

```
Train on 54000 samples, validate on 6000 samples
Epoch 1/20
54000/54000 [==============================] - 2s 45us/sample - loss: 0.5722 - accuracy: 0.8289 - val_loss:
0.1713 - val_accuracy: 0.9543
Epoch 2/20
54000/54000 [==============================] - 2s 36us/sample - loss: 0.2086 - accuracy: 0.9440 - val_loss:
0.1236 - val_accuracy: 0.9645
Epoch 3/20
54000/54000 [==============================] - 2s 37us/sample - loss: 0.1477 - accuracy: 0.9596 - val_loss:
0.1111 - val_accuracy: 0.9687
Epoch 4/20
54000/54000 [==============================] - 2s 35us/sample - loss: 0.1118 - accuracy: 0.9696 - val_loss:
0.1039 - val_accuracy: 0.9728
Epoch 5/20
54000/54000 [==============================] - 2s 34us/sample - loss: 0.0938 - accuracy: 0.9745 - val_loss:
0.0951 - val_accuracy: 0.9752
Epoch 6/20
54000/54000 [==============================] - 2s 35us/sample - loss: 0.0754 - accuracy: 0.9796 - val_loss:
0.0917 - val_accuracy: 0.9767
Epoch 7/20
54000/54000 [==============================] - 2s 35us/sample - loss: 0.0636 - accuracy: 0.9829 - val_loss:
0.0865 - val_accuracy: 0.9770
Epoch 8/20
54000/54000 [==============================] - 2s 38us/sample - loss: 0.0510 - accuracy: 0.9852 - val_loss:
0.0865 - val_accuracy: 0.9777
Epoch 9/20
54000/54000 [==============================] - 2s 36us/sample - loss: 0.0434 - accuracy: 0.9878 - val_loss:
0.0967 - val_accuracy: 0.9763
Epoch 10/20
54000/54000 [==============================] - 2s 36us/sample - loss: 0.0412 - accuracy: 0.9887 - val_loss:
0.0955 - val_accuracy: 0.9778
Epoch 11/20
54000/54000 [==============================] - 2s 35us/sample - loss: 0.0363 - accuracy: 0.9897 - val_loss:
0.1028 - val_accuracy: 0.9780
Epoch 12/20
54000/54000 [==============================] - 2s 36us/sample - loss: 0.0321 - accuracy: 0.9906 - val_loss:
0.1033 - val_accuracy: 0.9748
Epoch 13/20
54000/54000 [==============================] - 2s 36us/sample - loss: 0.0264 - accuracy: 0.9926 - val_loss:
0.0940 - val_accuracy: 0.9787
Epoch 14/20
54000/54000 [==============================] - 2s 36us/sample - loss: 0.0239 - accuracy: 0.9930 - val_loss:
0.1132 - val_accuracy: 0.9753
```

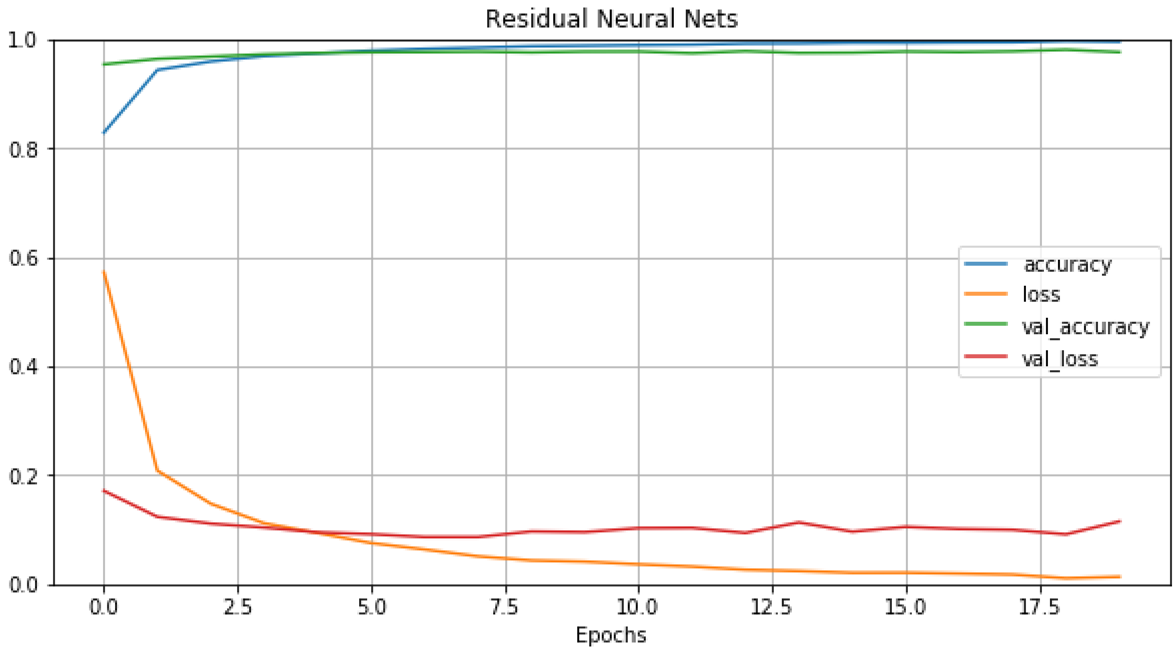
```
Epoch 15/20
54000/54000 [==============================] - 2s 37us/sample - loss: 0.0207 - accuracy: 0.9942 - val_loss:
0.0963 - val_accuracy: 0.9758
Epoch 16/20
54000/54000 [==============================] - 2s 38us/sample - loss: 0.0208 - accuracy: 0.9941 - val_loss:
0.1051 - val_accuracy: 0.9780
Epoch 17/20
54000/54000 [==============================] - 2s 37us/sample - loss: 0.0193 - accuracy: 0.9947 - val_loss:
0.1013 - val_accuracy: 0.9770
Epoch 18/20
54000/54000 [==============================] - 2s 35us/sample - loss: 0.0175 - accuracy: 0.9950 - val_loss:
0.0998 - val_accuracy: 0.9783
Epoch 19/20
54000/54000 [==============================] - 2s 35us/sample - loss: 0.0107 - accuracy: 0.9972 - val_loss:
0.0914 - val_accuracy: 0.9813
Epoch 20/20
54000/54000 [==============================] - 2s 36us/sample - loss: 0.0133 - accuracy: 0.9960 - val_loss:
0.1152 - val_accuracy: 0.9768
```

Out[6]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x26dbf270b00>
```

Out[6]:
```
(0, 1)
```

Out[6]:
```
Text(0.5, 0, 'Epochs')
```

Out[6]:
```
Text(0.5, 1.0, 'Residual Neural Nets')
```




```
10000/10000 [==============================] - 0s 33us/sample - loss: 0.1320 - accuracy: 0.9724
```

Out[6]:
```
[0.132009984324011, 0.9724]
```

```
Model: "resnet"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
img (InputLayer)             [(None, 784)]             0         
_________________________________________________________________
dense_8 (Dense)              (None, 128)               100480    
_________________________________________________________________
dense_9 (Dense)              (None, 128)               16512     
_________________________________________________________________
dense_10 (Dense)             (None, 128)               16512     
_________________________________________________________________
batch_normalization (BatchNo (None, 128)               512       
_________________________________________________________________
activation (Activation)      (None, 128)               0         
_________________________________________________________________
dropout (Dropout)            (None, 128)               0         
_________________________________________________________________
dense_11 (Dense)             (None, 128)               16512     
_________________________________________________________________
batch_normalization_1 (Batch (None, 128)               512       
_________________________________________________________________
activation_1 (Activation)    (None, 128)               0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_12 (Dense)             (None, 10)                1290      
=================================================================
Total params: 152,330
Trainable params: 151,818
Non-trainable params: 512
_________________________________________________________________
```

Analysis:

- Accuracy: There is an increase in the training and validation accuracies.
- Speed: There has been slight increase in training speeds. Could be because of the complex nature of ResNets

Step 6: Now lets make a deeper ResNet. Make A network with 10 Residual Blocks.

In [7]:
```
# Create the deep ResNet here.
inputs = tf.keras.Input(shape=(784,), name='img')
x = Dense(128, activation='relu')(inputs)
block_1_output = Dense(128, activation='relu')(x)

x = Dense(128)(block_1_output)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)
x = Dense(128)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)
block_2_output = tf.keras.layers.add([x, block_1_output])

# We will repeat above for as many times as we think our computer can handle. For now lets make just three Re
sidual Blocks.
x = Dense(128)(block_2_output)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)
x = Dense(128)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)
block_3_output = tf.keras.layers.add([x, block_2_output])

x = Dense(128)(block_3_output)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)
x = Dense(128)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)
block_4_output = tf.keras.layers.add([x, block_3_output])

x = Dense(128)(block_4_output)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)
x = Dense(128)(x)
```

```
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)
block_5_output = tf.keras.layers.add([x, block_4_output])

x = Dense(128)(block_5_output)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)
x = Dense(128)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)
block_6_output = tf.keras.layers.add([x, block_5_output])

x = Dense(128)(block_6_output)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)
x = Dense(128)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)
block_7_output = tf.keras.layers.add([x, block_6_output])

x = Dense(128)(block_7_output)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)
x = Dense(128)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)
block_8_output = tf.keras.layers.add([x, block_7_output])

x = Dense(128)(block_8_output)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)
x = Dense(128)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)
```

```
block_9_output = tf.keras.layers.add([x, block_8_output])

x = Dense(128)(block_9_output)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)
x = Dense(128)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)
block_10_output = tf.keras.layers.add([x, block_9_output])

# Now we cast the final Residual output into a dense layer to be able to classify the output easier.
x = Dense(128, activation='relu')(block_10_output)
x = Dropout(0.5)(x)

outputs = Dense(10, activation='softmax')(x)

r10model = tf.keras.Model(inputs, outputs, name='resnet')

r10model.compile(Adam(amsgrad=True), 'sparse_categorical_crossentropy', metrics=['accuracy'])

mx_train = mx_train.reshape(60000, 784)
mx_test = mx_test.reshape(10000, 784)
#mx_train = mx_train.astype('float32')
#mx_test = mx_test.astype('float32')

r10model_detail = r10model.fit(mx_train, my_train,
              batch_size=128,
              epochs=20,
              validation_split=0.1)

#plot accuracies for each epoch
r10history = pd.DataFrame(r10model_detail.history)
r10history.plot(figsize=(10,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.xlabel('Epochs')
plt.title('Residual Neural Nets')
plt.show()

#After training the model, evaluate the test set
r10model.evaluate(mx_test,my_test)
```

```
#Print the summary of the model
r10model.summary()
```

```
Train on 54000 samples, validate on 6000 samples
Epoch 1/20
54000/54000 [==============================] - 12s 231us/sample - loss: 2.1663 - accuracy: 0.2324 - val_loss:
1.7797 - val_accuracy: 0.2842
Epoch 2/20
54000/54000 [==============================] - 10s 176us/sample - loss: 1.1125 - accuracy: 0.6068 - val_loss:
0.4452 - val_accuracy: 0.8578
Epoch 3/20
54000/54000 [==============================] - 10s 178us/sample - loss: 0.5670 - accuracy: 0.8381 - val_loss:
0.2244 - val_accuracy: 0.9340
Epoch 4/20
54000/54000 [==============================] - 10s 176us/sample - loss: 0.3522 - accuracy: 0.9106 - val_loss:
0.1545 - val_accuracy: 0.9572
Epoch 5/20
54000/54000 [==============================] - 10s 177us/sample - loss: 0.2654 - accuracy: 0.9353 - val_loss:
0.1361 - val_accuracy: 0.9613
Epoch 6/20
54000/54000 [==============================] - 9s 174us/sample - loss: 0.2094 - accuracy: 0.9476 - val_loss:
0.1236 - val_accuracy: 0.9655
Epoch 7/20
54000/54000 [==============================] - 9s 168us/sample - loss: 0.1783 - accuracy: 0.9543 - val_loss:
0.1064 - val_accuracy: 0.9708
Epoch 8/20
54000/54000 [==============================] - 9s 162us/sample - loss: 0.1516 - accuracy: 0.9612 - val_loss:
0.1044 - val_accuracy: 0.9725
Epoch 9/20
54000/54000 [==============================] - 9s 162us/sample - loss: 0.1353 - accuracy: 0.9652 - val_loss:
0.1006 - val_accuracy: 0.9722
Epoch 10/20
54000/54000 [==============================] - 9s 165us/sample - loss: 0.1180 - accuracy: 0.9698 - val_loss:
0.0970 - val_accuracy: 0.9733
Epoch 11/20
54000/54000 [==============================] - 9s 158us/sample - loss: 0.1073 - accuracy: 0.9724 - val_loss:
0.0950 - val_accuracy: 0.9742
Epoch 12/20
54000/54000 [==============================] - 9s 169us/sample - loss: 0.0985 - accuracy: 0.9749 - val_loss:
0.0955 - val_accuracy: 0.9762
Epoch 13/20
54000/54000 [==============================] - 8s 154us/sample - loss: 0.0851 - accuracy: 0.9774 - val_loss:
0.1015 - val_accuracy: 0.9737
Epoch 14/20
54000/54000 [==============================] - 9s 170us/sample - loss: 0.0826 - accuracy: 0.9780 - val_loss:
0.0949 - val_accuracy: 0.9780
```

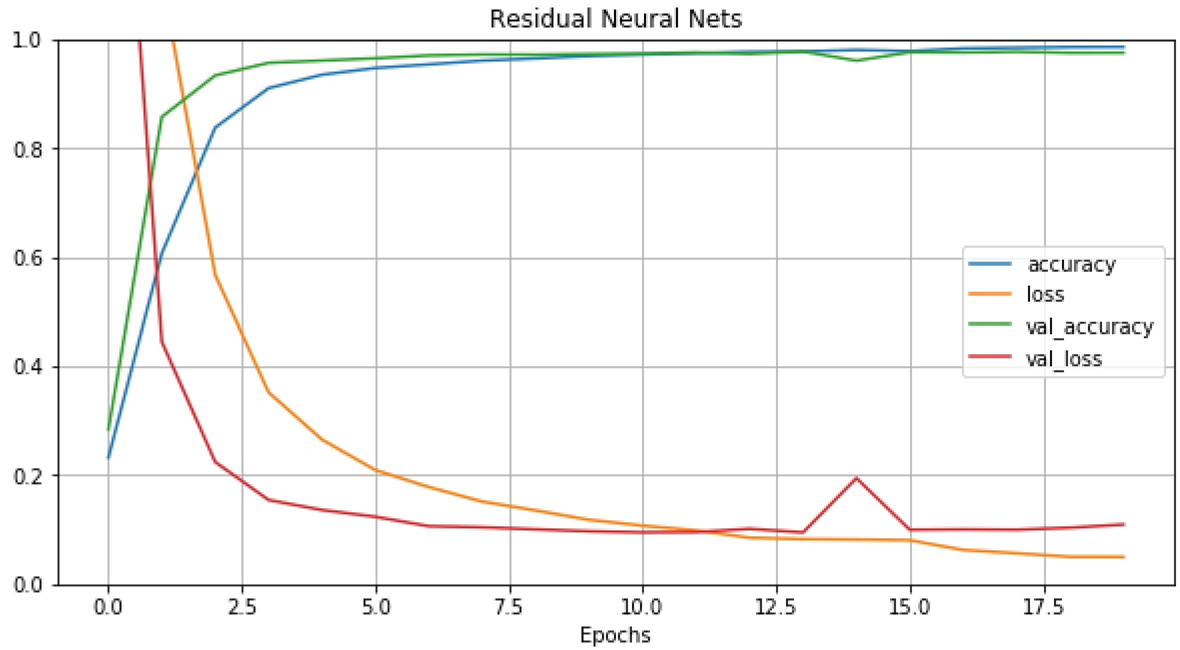
```
Epoch 15/20
54000/54000 [==============================] - 8s 154us/sample - loss: 0.0818 - accuracy: 0.9809 - val_loss:
0.1946 - val_accuracy: 0.9612
Epoch 16/20
54000/54000 [==============================] - 9s 168us/sample - loss: 0.0805 - accuracy: 0.9787 - val_loss:
0.0997 - val_accuracy: 0.9767
Epoch 17/20
54000/54000 [==============================] - 9s 159us/sample - loss: 0.0626 - accuracy: 0.9838 - val_loss:
0.1005 - val_accuracy: 0.9762
Epoch 18/20
54000/54000 [==============================] - 9s 163us/sample - loss: 0.0566 - accuracy: 0.9850 - val_loss:
0.0998 - val_accuracy: 0.9777
Epoch 19/20
54000/54000 [==============================] - 9s 161us/sample - loss: 0.0501 - accuracy: 0.9861 - val_loss:
0.1035 - val_accuracy: 0.9755
Epoch 20/20
54000/54000 [==============================] - 9s 161us/sample - loss: 0.0498 - accuracy: 0.9864 - val_loss:
0.1094 - val_accuracy: 0.9757
```

Out[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x26dd9ca9b70>
```

Out[7]:
```
(0, 1)
```

Out[7]:
```
Text(0.5, 0, 'Epochs')
```

Out[7]:
```
Text(0.5, 1.0, 'Residual Neural Nets')
```




```
10000/10000 [==============================] - 1s 64us/sample - loss: 0.1344 - accuracy: 0.9719
```

Out[7]:
```
[0.13437970478771605, 0.9719]
```

```
Model: "resnet"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to                     
==================================================================================================
img (InputLayer)                [(None, 784)]        0                                            
__________________________________________________________________________________________________
dense_13 (Dense)                (None, 128)          100480      img[0][0]                        
__________________________________________________________________________________________________
dense_14 (Dense)                (None, 128)          16512       dense_13[0][0]                   
__________________________________________________________________________________________________
dense_15 (Dense)                (None, 128)          16512       dense_14[0][0]                   
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 128)          512         dense_15[0][0]                   
__________________________________________________________________________________________________
activation_2 (Activation)       (None, 128)          0           batch_normalization_2[0][0]      
__________________________________________________________________________________________________
dropout_2 (Dropout)             (None, 128)          0           activation_2[0][0]               
__________________________________________________________________________________________________
dense_16 (Dense)                (None, 128)          16512       dropout_2[0][0]                  
__________________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 128)          512         dense_16[0][0]                   
__________________________________________________________________________________________________
activation_3 (Activation)       (None, 128)          0           batch_normalization_3[0][0]      
__________________________________________________________________________________________________
dropout_3 (Dropout)             (None, 128)          0           activation_3[0][0]               
__________________________________________________________________________________________________
add (Add)                       (None, 128)          0           dropout_3[0][0]                  
                                                                 dense_14[0][0]                   
__________________________________________________________________________________________________
dense_17 (Dense)                (None, 128)          16512       add[0][0]                        
__________________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, 128)          512         dense_17[0][0]                   
__________________________________________________________________________________________________
activation_4 (Activation)       (None, 128)          0           batch_normalization_4[0][0]      
__________________________________________________________________________________________________
dropout_4 (Dropout)             (None, 128)          0           activation_4[0][0]               
__________________________________________________________________________________________________
dense_18 (Dense)                (None, 128)          16512       dropout_4[0][0]                  
__________________________________________________________________________________________________
batch_normalization_5 (BatchNor (None, 128)          512         dense_18[0][0]                   
__________________________________________________________________________________________________
activation_5 (Activation)       (None, 128)          0           batch_normalization_5[0][0]      
__________________________________________________________________________________________________
```

dropout_5 (Dropout)	(None, 128)	0	activation_5[0][0]
add_1 (Add)	(None, 128)	0	dropout_5[0][0] add[0][0]
dense_19 (Dense)	(None, 128)	16512	add_1[0][0]
batch_normalization_6 (BatchNor	(None, 128)	512	dense_19[0][0]
activation_6 (Activation)	(None, 128)	0	batch_normalization_6[0][0]
dropout_6 (Dropout)	(None, 128)	0	activation_6[0][0]
dense_20 (Dense)	(None, 128)	16512	dropout_6[0][0]
batch_normalization_7 (BatchNor	(None, 128)	512	dense_20[0][0]
activation_7 (Activation)	(None, 128)	0	batch_normalization_7[0][0]
dropout_7 (Dropout)	(None, 128)	0	activation_7[0][0]
add_2 (Add)	(None, 128)	0	dropout_7[0][0] add_1[0][0]
dense_21 (Dense)	(None, 128)	16512	add_2[0][0]
batch_normalization_8 (BatchNor	(None, 128)	512	dense_21[0][0]
activation_8 (Activation)	(None, 128)	0	batch_normalization_8[0][0]
dropout_8 (Dropout)	(None, 128)	0	activation_8[0][0]
dense_22 (Dense)	(None, 128)	16512	dropout_8[0][0]
batch_normalization_9 (BatchNor	(None, 128)	512	dense_22[0][0]
activation_9 (Activation)	(None, 128)	0	batch_normalization_9[0][0]
dropout_9 (Dropout)	(None, 128)	0	activation_9[0][0]
add_3 (Add)	(None, 128)	0	dropout_9[0][0] add_2[0][0]

```
dense_23 (Dense)                (None, 128)          16512       add_3[0][0]
__________________________________________________________________________________________________
batch_normalization_10 (BatchNo (None, 128)          512         dense_23[0][0]
__________________________________________________________________________________________________
activation_10 (Activation)      (None, 128)          0           batch_normalization_10[0][0]
__________________________________________________________________________________________________
dropout_10 (Dropout)            (None, 128)          0           activation_10[0][0]
__________________________________________________________________________________________________
dense_24 (Dense)                (None, 128)          16512       dropout_10[0][0]
__________________________________________________________________________________________________
batch_normalization_11 (BatchNo (None, 128)          512         dense_24[0][0]
__________________________________________________________________________________________________
activation_11 (Activation)      (None, 128)          0           batch_normalization_11[0][0]
__________________________________________________________________________________________________
dropout_11 (Dropout)            (None, 128)          0           activation_11[0][0]
__________________________________________________________________________________________________
add_4 (Add)                     (None, 128)          0           dropout_11[0][0]
                                                                 add_3[0][0]
__________________________________________________________________________________________________
dense_25 (Dense)                (None, 128)          16512       add_4[0][0]
__________________________________________________________________________________________________
batch_normalization_12 (BatchNo (None, 128)          512         dense_25[0][0]
__________________________________________________________________________________________________
activation_12 (Activation)      (None, 128)          0           batch_normalization_12[0][0]
__________________________________________________________________________________________________
dropout_12 (Dropout)            (None, 128)          0           activation_12[0][0]
__________________________________________________________________________________________________
dense_26 (Dense)                (None, 128)          16512       dropout_12[0][0]
__________________________________________________________________________________________________
batch_normalization_13 (BatchNo (None, 128)          512         dense_26[0][0]
__________________________________________________________________________________________________
activation_13 (Activation)      (None, 128)          0           batch_normalization_13[0][0]
__________________________________________________________________________________________________
dropout_13 (Dropout)            (None, 128)          0           activation_13[0][0]
__________________________________________________________________________________________________
add_5 (Add)                     (None, 128)          0           dropout_13[0][0]
                                                                 add_4[0][0]
__________________________________________________________________________________________________
dense_27 (Dense)                (None, 128)          16512       add_5[0][0]
__________________________________________________________________________________________________
batch_normalization_14 (BatchNo (None, 128)          512         dense_27[0][0]
__________________________________________________________________________________________________
activation_14 (Activation)      (None, 128)          0           batch_normalization_14[0][0]
```

```
dropout_14 (Dropout)           (None, 128)          0           activation_14[0][0]
__________________________________________________________________________________________________
dense_28 (Dense)               (None, 128)          16512       dropout_14[0][0]
__________________________________________________________________________________________________
batch_normalization_15 (BatchNo (None, 128)          512         dense_28[0][0]
__________________________________________________________________________________________________
activation_15 (Activation)     (None, 128)          0           batch_normalization_15[0][0]
__________________________________________________________________________________________________
dropout_15 (Dropout)           (None, 128)          0           activation_15[0][0]
__________________________________________________________________________________________________
add_6 (Add)                    (None, 128)          0           dropout_15[0][0]
                                                                add_5[0][0]
__________________________________________________________________________________________________
dense_29 (Dense)               (None, 128)          16512       add_6[0][0]
__________________________________________________________________________________________________
batch_normalization_16 (BatchNo (None, 128)          512         dense_29[0][0]
__________________________________________________________________________________________________
activation_16 (Activation)     (None, 128)          0           batch_normalization_16[0][0]
__________________________________________________________________________________________________
dropout_16 (Dropout)           (None, 128)          0           activation_16[0][0]
__________________________________________________________________________________________________
dense_30 (Dense)               (None, 128)          16512       dropout_16[0][0]
__________________________________________________________________________________________________
batch_normalization_17 (BatchNo (None, 128)          512         dense_30[0][0]
__________________________________________________________________________________________________
activation_17 (Activation)     (None, 128)          0           batch_normalization_17[0][0]
__________________________________________________________________________________________________
dropout_17 (Dropout)           (None, 128)          0           activation_17[0][0]
__________________________________________________________________________________________________
add_7 (Add)                    (None, 128)          0           dropout_17[0][0]
                                                                add_6[0][0]
__________________________________________________________________________________________________
dense_31 (Dense)               (None, 128)          16512       add_7[0][0]
__________________________________________________________________________________________________
batch_normalization_18 (BatchNo (None, 128)          512         dense_31[0][0]
__________________________________________________________________________________________________
activation_18 (Activation)     (None, 128)          0           batch_normalization_18[0][0]
__________________________________________________________________________________________________
dropout_18 (Dropout)           (None, 128)          0           activation_18[0][0]
__________________________________________________________________________________________________
dense_32 (Dense)               (None, 128)          16512       dropout_18[0][0]
__________________________________________________________________________________________________
```

```
batch_normalization_19 (BatchNo (None, 128)          512         dense_32[0][0]
__________________________________________________________________________________________________
activation_19 (Activation)      (None, 128)          0           batch_normalization_19[0][0]
__________________________________________________________________________________________________
dropout_19 (Dropout)            (None, 128)          0           activation_19[0][0]
__________________________________________________________________________________________________
add_8 (Add)                     (None, 128)          0           dropout_19[0][0]
                                                                 add_7[0][0]
__________________________________________________________________________________________________
dense_33 (Dense)                (None, 128)          16512       add_8[0][0]
__________________________________________________________________________________________________
dropout_20 (Dropout)            (None, 128)          0           dense_33[0][0]
__________________________________________________________________________________________________
dense_34 (Dense)                (None, 10)           1290        dropout_20[0][0]
==================================================================================================
Total params: 441,226
Trainable params: 436,618
Non-trainable params: 4,608
__________________________________________________________________________________________________
```

Analysis:

- Compared to Shallow net, there is no increase in the Accuracy, but training time is increased significantly.
- We can see, in summary, significant increase in number of parameters tuned.
- I think, we do not need any extra hidden layers to handle the complexity of MNIST data. Shallow ResNet would do better with less training time.

Step 7: ResNets that have one and three skip connections.

In [8]:
```
inputs = tf.keras.Input(shape=(784,), name='img')
x = Dense(128, activation='relu')(inputs)
block_1_output = Dense(128, activation='relu')(x)

# Here is a one skip connection block.
x = Dense(128)(block_1_output)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)
block_2_output = tf.keras.layers.add([x, block_1_output])

# Here is a three skip connection block.
x = Dense(128)(block_2_output)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)
x = Dense(128)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)
x = Dense(128)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)
block_3_output = tf.keras.layers.add([x, block_2_output])

x = Dense(128, activation='relu')(block_3_output)
x = Dropout(0.5)(x)
outputs = Dense(10, activation='softmax')(x)

r13model = tf.keras.Model(inputs, outputs, name='resnet')

r13model.compile(Adam(amsgrad=True), 'sparse_categorical_crossentropy', metrics=['accuracy'])

mx_train = mx_train.reshape(60000, 784)
mx_test = mx_test.reshape(10000, 784)
#mx_train = mx_train.astype('float32')
#mx_test = mx_test.astype('float32')

r13model_detail = r13model.fit(mx_train, my_train,
              batch_size=128,
              epochs=20,
```

```
              validation_split=0.1)

#plot accuracies for each epoch
r13history = pd.DataFrame(r13model_detail.history)
r13history.plot(figsize=(10,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.xlabel('Epochs')
plt.title('Residual Neural Nets')
plt.show()

#After training the model, evaluate the test set
r13model.evaluate(mx_test,my_test)

#Print the summary of the model
r13model.summary()
```

```
Train on 54000 samples, validate on 6000 samples
Epoch 1/20
54000/54000 [==============================] - 4s 72us/sample - loss: 0.8241 - accuracy: 0.7452 - val_loss:
0.1998 - val_accuracy: 0.9430
Epoch 2/20
54000/54000 [==============================] - 3s 58us/sample - loss: 0.2604 - accuracy: 0.9300 - val_loss:
0.1220 - val_accuracy: 0.9650
Epoch 3/20
54000/54000 [==============================] - 3s 58us/sample - loss: 0.1819 - accuracy: 0.9525 - val_loss:
0.1201 - val_accuracy: 0.9653
Epoch 4/20
54000/54000 [==============================] - 3s 58us/sample - loss: 0.1338 - accuracy: 0.9639 - val_loss:
0.1097 - val_accuracy: 0.9670
Epoch 5/20
54000/54000 [==============================] - 3s 57us/sample - loss: 0.1023 - accuracy: 0.9718 - val_loss:
0.0931 - val_accuracy: 0.9735
Epoch 6/20
54000/54000 [==============================] - 3s 57us/sample - loss: 0.0892 - accuracy: 0.9759 - val_loss:
0.0998 - val_accuracy: 0.9738
Epoch 7/20
54000/54000 [==============================] - 3s 57us/sample - loss: 0.0729 - accuracy: 0.9808 - val_loss:
0.0976 - val_accuracy: 0.9750
Epoch 8/20
54000/54000 [==============================] - 3s 58us/sample - loss: 0.0609 - accuracy: 0.9821 - val_loss:
0.1003 - val_accuracy: 0.9758
Epoch 9/20
54000/54000 [==============================] - 3s 59us/sample - loss: 0.0543 - accuracy: 0.9854 - val_loss:
0.1009 - val_accuracy: 0.9757
Epoch 10/20
54000/54000 [==============================] - 3s 58us/sample - loss: 0.0444 - accuracy: 0.9875 - val_loss:
0.0961 - val_accuracy: 0.9783
Epoch 11/20
54000/54000 [==============================] - 3s 58us/sample - loss: 0.0395 - accuracy: 0.9886 - val_loss:
0.1004 - val_accuracy: 0.9765
Epoch 12/20
54000/54000 [==============================] - 3s 57us/sample - loss: 0.0340 - accuracy: 0.9902 - val_loss:
0.0931 - val_accuracy: 0.9788
Epoch 13/20
54000/54000 [==============================] - 3s 58us/sample - loss: 0.0301 - accuracy: 0.9914 - val_loss:
0.0988 - val_accuracy: 0.9767
Epoch 14/20
54000/54000 [==============================] - 3s 58us/sample - loss: 0.0289 - accuracy: 0.9915 - val_loss:
0.1064 - val_accuracy: 0.9760
```

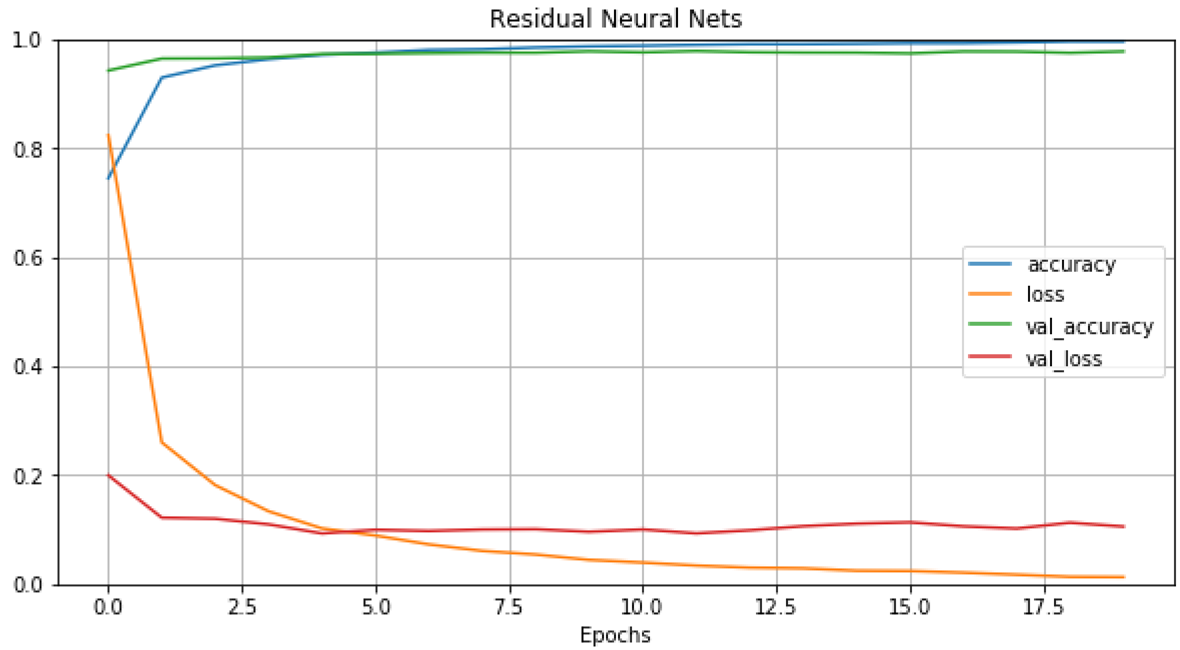
```
Epoch 15/20
54000/54000 [==============================] - 3s 57us/sample - loss: 0.0246 - accuracy: 0.9923 - val_loss: 0.1110 - val_accuracy: 0.9758
Epoch 16/20
54000/54000 [==============================] - 3s 57us/sample - loss: 0.0240 - accuracy: 0.9933 - val_loss: 0.1135 - val_accuracy: 0.9745
Epoch 17/20
54000/54000 [==============================] - 3s 57us/sample - loss: 0.0210 - accuracy: 0.9933 - val_loss: 0.1061 - val_accuracy: 0.9782
Epoch 18/20
54000/54000 [==============================] - 3s 59us/sample - loss: 0.0172 - accuracy: 0.9947 - val_loss: 0.1021 - val_accuracy: 0.9778
Epoch 19/20
54000/54000 [==============================] - 3s 59us/sample - loss: 0.0133 - accuracy: 0.9965 - val_loss: 0.1128 - val_accuracy: 0.9755
Epoch 20/20
54000/54000 [==============================] - 3s 58us/sample - loss: 0.0128 - accuracy: 0.9964 - val_loss: 0.1057 - val_accuracy: 0.9782
```

Out[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x26ddefe75c0>
```

Out[8]:
```
(0, 1)
```

Out[8]:
```
Text(0.5, 0, 'Epochs')
```

Out[8]:
```
Text(0.5, 1.0, 'Residual Neural Nets')
```




```
10000/10000 [==============================] - 0s 40us/sample - loss: 0.1323 - accuracy: 0.9757
```

Out[8]:
```
[0.13233828188165084, 0.9757]
```

```
Model: "resnet"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to                     
==================================================================================================
img (InputLayer)                [(None, 784)]        0                                            
__________________________________________________________________________________________________
dense_35 (Dense)                (None, 128)          100480      img[0][0]                        
__________________________________________________________________________________________________
dense_36 (Dense)                (None, 128)          16512       dense_35[0][0]                   
__________________________________________________________________________________________________
dense_37 (Dense)                (None, 128)          16512       dense_36[0][0]                   
__________________________________________________________________________________________________
batch_normalization_20 (BatchNo (None, 128)          512         dense_37[0][0]                   
__________________________________________________________________________________________________
activation_20 (Activation)      (None, 128)          0           batch_normalization_20[0][0]     
__________________________________________________________________________________________________
dropout_21 (Dropout)            (None, 128)          0           activation_20[0][0]              
__________________________________________________________________________________________________
add_9 (Add)                     (None, 128)          0           dropout_21[0][0]                 
                                                                 dense_36[0][0]                   
__________________________________________________________________________________________________
dense_38 (Dense)                (None, 128)          16512       add_9[0][0]                      
__________________________________________________________________________________________________
batch_normalization_21 (BatchNo (None, 128)          512         dense_38[0][0]                   
__________________________________________________________________________________________________
activation_21 (Activation)      (None, 128)          0           batch_normalization_21[0][0]     
__________________________________________________________________________________________________
dropout_22 (Dropout)            (None, 128)          0           activation_21[0][0]              
__________________________________________________________________________________________________
dense_39 (Dense)                (None, 128)          16512       dropout_22[0][0]                 
__________________________________________________________________________________________________
batch_normalization_22 (BatchNo (None, 128)          512         dense_39[0][0]                   
__________________________________________________________________________________________________
activation_22 (Activation)      (None, 128)          0           batch_normalization_22[0][0]     
__________________________________________________________________________________________________
dropout_23 (Dropout)            (None, 128)          0           activation_22[0][0]              
__________________________________________________________________________________________________
dense_40 (Dense)                (None, 128)          16512       dropout_23[0][0]                 
__________________________________________________________________________________________________
batch_normalization_23 (BatchNo (None, 128)          512         dense_40[0][0]                   
__________________________________________________________________________________________________
activation_23 (Activation)      (None, 128)          0           batch_normalization_23[0][0]     
__________________________________________________________________________________________________
```

```
dropout_24 (Dropout)            (None, 128)          0           activation_23[0][0]
________________________________________________________________________________________________
add_10 (Add)                    (None, 128)          0           dropout_24[0][0]
                                                                 add_9[0][0]
________________________________________________________________________________________________
dense_41 (Dense)                (None, 128)          16512       add_10[0][0]
________________________________________________________________________________________________
dropout_25 (Dropout)            (None, 128)          0           dense_41[0][0]
________________________________________________________________________________________________
dense_42 (Dense)                (None, 10)           1290        dropout_25[0][0]
================================================================================================
Total params: 202,890
Trainable params: 201,866
Non-trainable params: 1,024
________________________________________________________________________________________________
```

Analysis:

- Training speed is improved compared to 10 resnets.
- Accuracy remained same. I think for MNIST data, we can get good accuracy with less number of trainable layers.

Step 8: Functional Model for CIFAR with the same baseline as MNIST.

In [21]:
```
# Create the CIFAR baseline here.
# Create the functional Baseline here.
input_layer = keras.layers.Input(shape=cx_train.shape[1:])
il = Flatten()(input_layer)
h1 = keras.layers.Dense(64,activation="relu")(il)
h2 = keras.layers.Dense(64,activation="relu")(h1)
h3 = keras.layers.Dense(64,activation="relu")(h2)
output_layer = keras.layers.Dense(10, activation="softmax")(h3)
fcmodel = keras.models.Model(inputs=[input_layer], outputs=[output_layer])

fcmodel.compile(loss='categorical_crossentropy',
              optimizer=keras.optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True),metrics=['accu
racy'])

# Convert class vectors to binary class matrices.
cy_train = keras.utils.to_categorical(cy_train, 10)
cy_test = keras.utils.to_categorical(cy_test, 10)


fcmodel_detail = fcmodel.fit(cx_train, cy_train, epochs=20, batch_size=64, validation_split=0.1)

#plot accuracies for each epoch
fchistory = pd.DataFrame(fcmodel_detail.history)
fchistory.plot(figsize=(10,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.xlabel('Epochs')
plt.title('Functional Neural Nets')
plt.show()

#After training the model, evaluate the test set
fcmodel.evaluate(cx_test,cy_test)

#Print the summary of the model
fcmodel.summary()
```

```
Train on 45000 samples, validate on 5000 samples
Epoch 1/20
45000/45000 [==============================] - 2s 49us/sample - loss: 2.0551 - accuracy: 0.2518 - val_loss:
2.0192 - val_accuracy: 0.2530
Epoch 2/20
45000/45000 [==============================] - 2s 45us/sample - loss: 1.8717 - accuracy: 0.3299 - val_loss:
1.9179 - val_accuracy: 0.3186
Epoch 3/20
45000/45000 [==============================] - 2s 45us/sample - loss: 1.7888 - accuracy: 0.3640 - val_loss:
1.7789 - val_accuracy: 0.3674
Epoch 4/20
45000/45000 [==============================] - 2s 45us/sample - loss: 1.7383 - accuracy: 0.3819 - val_loss:
1.7761 - val_accuracy: 0.3690
Epoch 5/20
45000/45000 [==============================] - 2s 44us/sample - loss: 1.7014 - accuracy: 0.3942 - val_loss:
1.7782 - val_accuracy: 0.3620
Epoch 6/20
45000/45000 [==============================] - 2s 44us/sample - loss: 1.6642 - accuracy: 0.4099 - val_loss:
1.7122 - val_accuracy: 0.3922
Epoch 7/20
45000/45000 [==============================] - 2s 45us/sample - loss: 1.6336 - accuracy: 0.4206 - val_loss:
1.6767 - val_accuracy: 0.4076
Epoch 8/20
45000/45000 [==============================] - 2s 44us/sample - loss: 1.6056 - accuracy: 0.4280 - val_loss:
1.7612 - val_accuracy: 0.3822
Epoch 9/20
45000/45000 [==============================] - 2s 44us/sample - loss: 1.5821 - accuracy: 0.4354 - val_loss:
1.7629 - val_accuracy: 0.3720
Epoch 10/20
45000/45000 [==============================] - 2s 45us/sample - loss: 1.5592 - accuracy: 0.4449 - val_loss:
1.7059 - val_accuracy: 0.3902
Epoch 11/20
45000/45000 [==============================] - 2s 45us/sample - loss: 1.5419 - accuracy: 0.4524 - val_loss:
1.6567 - val_accuracy: 0.4200
Epoch 12/20
45000/45000 [==============================] - 2s 44us/sample - loss: 1.5217 - accuracy: 0.4564 - val_loss:
1.6910 - val_accuracy: 0.4006
Epoch 13/20
45000/45000 [==============================] - 2s 44us/sample - loss: 1.5031 - accuracy: 0.4665 - val_loss:
1.6142 - val_accuracy: 0.4370
Epoch 14/20
45000/45000 [==============================] - 2s 45us/sample - loss: 1.4863 - accuracy: 0.4712 - val_loss:
1.6621 - val_accuracy: 0.4128
```

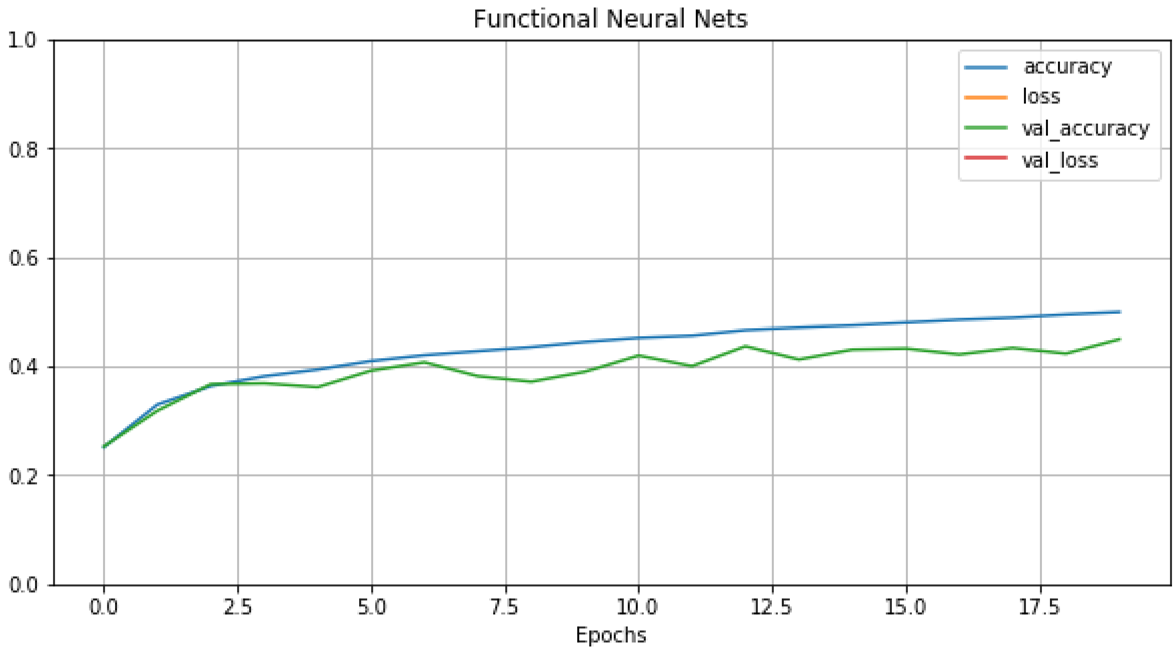
```
Epoch 15/20
45000/45000 [==============================] - 2s 44us/sample - loss: 1.4744 - accuracy: 0.4751 - val_loss:
1.6356 - val_accuracy: 0.4308
Epoch 16/20
45000/45000 [==============================] - 2s 44us/sample - loss: 1.4618 - accuracy: 0.4801 - val_loss:
1.6317 - val_accuracy: 0.4328
Epoch 17/20
45000/45000 [==============================] - 2s 45us/sample - loss: 1.4453 - accuracy: 0.4852 - val_loss:
1.6374 - val_accuracy: 0.4220
Epoch 18/20
45000/45000 [==============================] - 2s 45us/sample - loss: 1.4319 - accuracy: 0.4888 - val_loss:
1.6045 - val_accuracy: 0.4340
Epoch 19/20
45000/45000 [==============================] - 2s 45us/sample - loss: 1.4184 - accuracy: 0.4946 - val_loss:
1.6605 - val_accuracy: 0.4236
Epoch 20/20
45000/45000 [==============================] - 2s 45us/sample - loss: 1.4029 - accuracy: 0.4992 - val_loss:
1.5867 - val_accuracy: 0.4498
```

```
Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x26dc0b11240>
```

```
Out[21]: (0, 1)
```

```
Out[21]: Text(0.5, 0, 'Epochs')
```

```
Out[21]: Text(0.5, 1.0, 'Functional Neural Nets')
```




```
10000/10000 [==============================] - 0s 36us/sample - loss: 1.6179 - accuracy: 0.4300
```

Out[21]:
```
[1.6179374866485596, 0.43]
```

```
Model: "model_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_7 (InputLayer)         [(None, 32, 32, 3)]       0
_________________________________________________________________
flatten_7 (Flatten)          (None, 3072)              0
_________________________________________________________________
dense_63 (Dense)             (None, 64)                196672
_________________________________________________________________
dense_64 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_65 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_66 (Dense)             (None, 10)                650
=================================================================
Total params: 205,642
Trainable params: 205,642
Non-trainable params: 0
_________________________________________________________________
```

Analysis:

- Training speed is same for both the data datasets
- Accuracy is reduced significantly. I think, Cifar data has RGB layered images and the domain knowledge of the dataset is complex. Funtional neural network is not able to understand the complexity of the dataset which resulted in poor accuracies.

Step 9: Shallow ResNet for CIFAR with the same layers and compare it with MNIST.

In [55]:

```
# Create the CIFAR ResNet here.
inputs = tf.keras.Input(shape=cx_train.shape[1:], name='img')
x = Dense(128, activation='relu')(inputs)
block_1_output = Dense(128, activation='relu')(x)

x = Dense(128)(block_1_output)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)
x = Dense(128)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)
y = Flatten()(x)
outputs = Dense(10, activation='softmax')(y)

rcmodel = tf.keras.Model(inputs, outputs, name='resnet')

rcmodel.compile(Adam(amsgrad=True), 'categorical_crossentropy', metrics=['accuracy'])

# Convert class vectors to binary class matrices.
cy_train = keras.utils.to_categorical(cy_train, 10)
cy_test = keras.utils.to_categorical(cy_test, 10)

rcmodel_detail = rcmodel.fit(cx_train, cy_train,
              batch_size=128,
              epochs=20,
              validation_split=0.1)

#plot accuracies for each epoch
rchistory = pd.DataFrame(rcmodel_detail.history)
rchistory.plot(figsize=(10,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.xlabel('Epochs')
plt.title('Residual Neural Nets')
plt.show()

#After training the model, evaluate the test set
rcmodel.evaluate(cx_test,cy_test)
```

```
#Print the summary of the model
rcmodel.summary()
```

```
Train on 45000 samples, validate on 5000 samples
Epoch 1/20
45000/45000 [==============================] - 330s 7ms/sample - loss: 1.7810 - accuracy: 0.3831 - val_loss:
7.7729 - val_accuracy: 0.0976
Epoch 2/20
45000/45000 [==============================] - 358s 8ms/sample - loss: 1.5499 - accuracy: 0.4672 - val_loss:
99.8459 - val_accuracy: 0.0976
Epoch 3/20
45000/45000 [==============================] - 314s 7ms/sample - loss: 1.4754 - accuracy: 0.4922 - val_loss:
3827.4924 - val_accuracy: 0.0958
Epoch 4/20
45000/45000 [==============================] - 314s 7ms/sample - loss: 17.9298 - accuracy: 0.3428 - val_loss:
102263.6198 - val_accuracy: 0.0950
Epoch 5/20
45000/45000 [==============================] - 351s 8ms/sample - loss: 10.6031 - accuracy: 0.2106 - val_loss:
17954.0223 - val_accuracy: 0.0958
Epoch 6/20
45000/45000 [==============================] - 363s 8ms/sample - loss: 6.3705 - accuracy: 0.2220 - val_loss:
11470.1549 - val_accuracy: 0.0958
Epoch 7/20
45000/45000 [==============================] - 313s 7ms/sample - loss: 4.3110 - accuracy: 0.2392 - val_loss:
5624.3404 - val_accuracy: 0.0958
Epoch 8/20
45000/45000 [==============================] - 339s 8ms/sample - loss: 3.4554 - accuracy: 0.2426 - val_loss:
3278.4047 - val_accuracy: 0.0958
Epoch 9/20
45000/45000 [==============================] - 351s 8ms/sample - loss: 2.9452 - accuracy: 0.2556 - val_loss:
1714.8383 - val_accuracy: 0.0958
Epoch 10/20
45000/45000 [==============================] - 349s 8ms/sample - loss: 2.4442 - accuracy: 0.2737 - val_loss:
767.6519 - val_accuracy: 0.0958
Epoch 11/20
45000/45000 [==============================] - 313s 7ms/sample - loss: 2.2266 - accuracy: 0.2822 - val_loss:
1975.6464 - val_accuracy: 0.0950
Epoch 12/20
45000/45000 [==============================] - 314s 7ms/sample - loss: 2.0512 - accuracy: 0.2875 - val_loss:
3693.7662 - val_accuracy: 0.0958
Epoch 13/20
45000/45000 [==============================] - 311s 7ms/sample - loss: 1.9472 - accuracy: 0.3235 - val_loss:
88.7906 - val_accuracy: 0.0990
Epoch 14/20
45000/45000 [==============================] - 564s 13ms/sample - loss: 1.9301 - accuracy: 0.3279 - val_loss:
5.8565 - val_accuracy: 0.0914
```

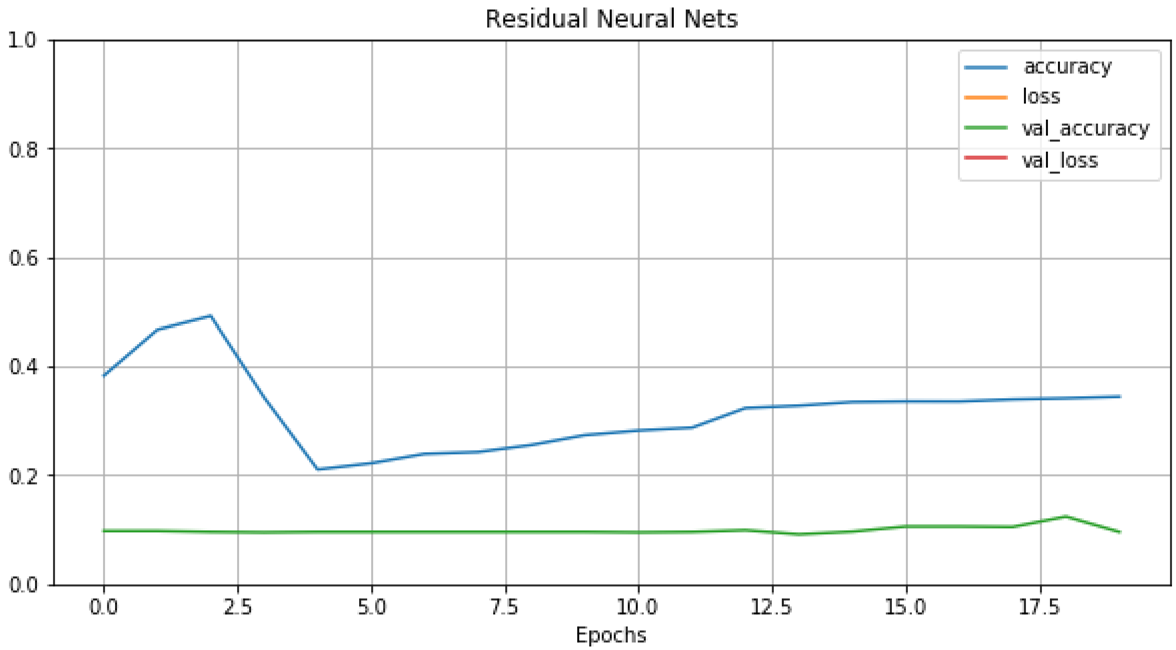
```
Epoch 15/20
45000/45000 [==============================] - 336s 7ms/sample - loss: 1.9207 - accuracy: 0.3346 - val_loss:
16.4478 - val_accuracy: 0.0964
Epoch 16/20
45000/45000 [==============================] - 323s 7ms/sample - loss: 1.9106 - accuracy: 0.3357 - val_loss:
246.8435 - val_accuracy: 0.1058
Epoch 17/20
45000/45000 [==============================] - 326s 7ms/sample - loss: 1.9117 - accuracy: 0.3356 - val_loss:
291.4979 - val_accuracy: 0.1058
Epoch 18/20
45000/45000 [==============================] - 324s 7ms/sample - loss: 1.9022 - accuracy: 0.3394 - val_loss:
3.0542 - val_accuracy: 0.1050
Epoch 19/20
45000/45000 [==============================] - 318s 7ms/sample - loss: 1.9001 - accuracy: 0.3415 - val_loss:
35.9115 - val_accuracy: 0.1240
Epoch 20/20
45000/45000 [==============================] - 325s 7ms/sample - loss: 1.8951 - accuracy: 0.3444 - val_loss:
24.2779 - val_accuracy: 0.0958
```

```
Out[55]: <matplotlib.axes._subplots.AxesSubplot at 0x26dfa32c0b8>

Out[55]: (0, 1)

Out[55]: Text(0.5, 0, 'Epochs')

Out[55]: Text(0.5, 1.0, 'Residual Neural Nets')
```




```
10000/10000 [==============================] - 10s 1ms/sample - loss: 24.1797 - accuracy: 0.10003s - loss: 2
4.0904 - accuracy: 0 - ETA: 3s - loss: 24. - ETA: 2s - loss: 24.0850 - accura - ETA: 1s - loss: 24.147
```

Out[55]:
```
[24.179732403564454, 0.1]
```

```
Model: "resnet"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
img (InputLayer)             [(None, 32, 32, 3)]       0         
_________________________________________________________________
dense_134 (Dense)            (None, 32, 32, 128)       512       
_________________________________________________________________
dense_135 (Dense)            (None, 32, 32, 128)       16512     
_________________________________________________________________
dense_136 (Dense)            (None, 32, 32, 128)       16512     
_________________________________________________________________
batch_normalization_60 (Batc (None, 32, 32, 128)       512       
_________________________________________________________________
activation_60 (Activation)   (None, 32, 32, 128)       0         
_________________________________________________________________
dropout_63 (Dropout)         (None, 32, 32, 128)       0         
_________________________________________________________________
dense_137 (Dense)            (None, 32, 32, 128)       16512     
_________________________________________________________________
batch_normalization_61 (Batc (None, 32, 32, 128)       512       
_________________________________________________________________
activation_61 (Activation)   (None, 32, 32, 128)       0         
_________________________________________________________________
dropout_64 (Dropout)         (None, 32, 32, 128)       0         
_________________________________________________________________
flatten_17 (Flatten)         (None, 131072)            0         
_________________________________________________________________
dense_138 (Dense)            (None, 10)                1310730   
=================================================================
Total params: 1,361,802
Trainable params: 1,361,290
Non-trainable params: 512
_________________________________________________________________
```

Step 10: Custom ResNet.

Change the layer widths, dropout layers, batch sizes, and skip connections to see what we could do to make the ResNet better. Use the knowledge you gained from Lab 3 to do this.

In [50]:
```
# Create the CIFAR ResNet here.
inputs = tf.keras.Input(shape=cx_train.shape[1:], name='img')
x = Dense(128, activation='relu')(inputs)
block_1_output = Dense(128, activation='relu')(x)

x = Dense(128)(block_1_output)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)
x = Dense(128)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)
y = Flatten()(x)
outputs = Dense(10, activation='softmax')(y)
rc10model = tf.keras.Model(inputs, outputs, name='resnet')

rc10model.compile(optimizer=Adam(learning_rate=lr_schedule(0)), loss='categorical_crossentropy', metrics=['ac
curacy'])

# Convert class vectors to binary class matrices.
cy_train = keras.utils.to_categorical(cy_train, 10)
cy_test = keras.utils.to_categorical(cy_test, 10)

lr_scheduler = LearningRateScheduler(lr_schedule)

lr_reducer = ReduceLROnPlateau(factor=np.sqrt(0.1),
                               cooldown=0,
                               patience=5,
                               min_lr=0.5e-6)

callbacks = [lr_reducer, lr_scheduler]

rc10model_detail = rc10model.fit(cx_train, cy_train,
              batch_size=32,
              epochs=30,
              validation_split=0.1,
              shuffle=True,
              callbacks=callbacks)

#plot accuracies for each epoch
rc10history = pd.DataFrame(rc10model_detail.history)
```

```
rc10history.plot(figsize=(10,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.xlabel('Epochs')
plt.title('Residual Neural Nets')
plt.show()

#After training the model, evaluate the test set
rc10model.evaluate(cx_test,cy_test)

#Print the summary of the model
rc10model.summary()
```

```
Learning rate:  0.001
Train on 45000 samples, validate on 5000 samples
Learning rate:  0.001
Epoch 1/30
45000/45000 [==============================] - 298s 7ms/sample - loss: 1.8943 - accuracy: 0.3394 - val_loss:
596.7904 - val_accuracy: 0.0958
Learning rate:  0.001
Epoch 2/30
45000/45000 [==============================] - 312s 7ms/sample - loss: 1.6885 - accuracy: 0.4205 - val_loss:
519.7225 - val_accuracy: 0.1070
Learning rate:  0.001
Epoch 3/30
45000/45000 [==============================] - 302s 7ms/sample - loss: 1.6079 - accuracy: 0.4495 - val_loss:
343.1269 - val_accuracy: 0.0958
Learning rate:  0.001
Epoch 4/30
45000/45000 [==============================] - 303s 7ms/sample - loss: 1.5752 - accuracy: 0.4606 - val_loss:
18261.2324 - val_accuracy: 0.0958
Learning rate:  0.001
Epoch 5/30
45000/45000 [==============================] - 295s 7ms/sample - loss: 8.9512 - accuracy: 0.2420 - val_loss:
625.8508 - val_accuracy: 0.0958
Learning rate:  0.001
Epoch 6/30
45000/45000 [==============================] - 296s 7ms/sample - loss: 1.9992 - accuracy: 0.2983 - val_loss:
1405.4320 - val_accuracy: 0.1038
Learning rate:  0.001
Epoch 7/30
45000/45000 [==============================] - 296s 7ms/sample - loss: 1.9330 - accuracy: 0.3268 - val_loss:
928.6470 - val_accuracy: 0.0958
Learning rate:  0.001
Epoch 8/30
45000/45000 [==============================] - 299s 7ms/sample - loss: 1.9090 - accuracy: 0.3355 - val_loss:
191.2962 - val_accuracy: 0.0958
Learning rate:  0.001
Epoch 9/30
45000/45000 [==============================] - 297s 7ms/sample - loss: 1.9124 - accuracy: 0.3365 - val_loss:
3226.9604 - val_accuracy: 0.1038
Learning rate:  0.001
Epoch 10/30
45000/45000 [==============================] - 305s 7ms/sample - loss: 1.8875 - accuracy: 0.3457 - val_loss:
1309.1224 - val_accuracy: 0.1050
Learning rate:  0.001
```

```
Epoch 11/30
45000/45000 [==============================] - 326s 7ms/sample - loss: 1.8754 - accuracy: 0.3454 - val_loss:
1488.5722 - val_accuracy: 0.0940
Learning rate:  0.0001
Epoch 12/30
45000/45000 [==============================] - 352s 8ms/sample - loss: 1.8525 - accuracy: 0.3561 - val_loss:
172.8564 - val_accuracy: 0.0984
Learning rate:  0.0001
Epoch 13/30
45000/45000 [==============================] - 361s 8ms/sample - loss: 1.8458 - accuracy: 0.3571 - val_loss:
91.9831 - val_accuracy: 0.1462
Learning rate:  0.0001
Epoch 14/30
45000/45000 [==============================] - 313s 7ms/sample - loss: 1.8418 - accuracy: 0.3612 - val_loss:
795.6470 - val_accuracy: 0.0986
Learning rate:  0.0001
Epoch 15/30
45000/45000 [==============================] - 296s 7ms/sample - loss: 1.8421 - accuracy: 0.3583 - val_loss:
284.3303 - val_accuracy: 0.1164
Learning rate:  0.0001
Epoch 16/30
45000/45000 [==============================] - 310s 7ms/sample - loss: 1.8282 - accuracy: 0.3646 - val_loss:
110.1915 - val_accuracy: 0.1152
Learning rate:  1e-05
Epoch 17/30
45000/45000 [==============================] - 337s 7ms/sample - loss: 1.8205 - accuracy: 0.3701 - val_loss:
176.8868 - val_accuracy: 0.1066
Learning rate:  1e-05
Epoch 18/30
45000/45000 [==============================] - 323s 7ms/sample - loss: 1.8184 - accuracy: 0.3675 - val_loss:
12.8034 - val_accuracy: 0.1542
Learning rate:  1e-05
Epoch 19/30
45000/45000 [==============================] - 339s 8ms/sample - loss: 1.8143 - accuracy: 0.3686 - val_loss:
56.8887 - val_accuracy: 0.1138
Learning rate:  1e-05
Epoch 20/30
45000/45000 [==============================] - 314s 7ms/sample - loss: 1.8162 - accuracy: 0.3681 - val_loss:
19.2211 - val_accuracy: 0.1254
Learning rate:  1e-05
Epoch 21/30
45000/45000 [==============================] - 344s 8ms/sample - loss: 1.8177 - accuracy: 0.3675 - val_loss:
32.0102 - val_accuracy: 0.1648
```

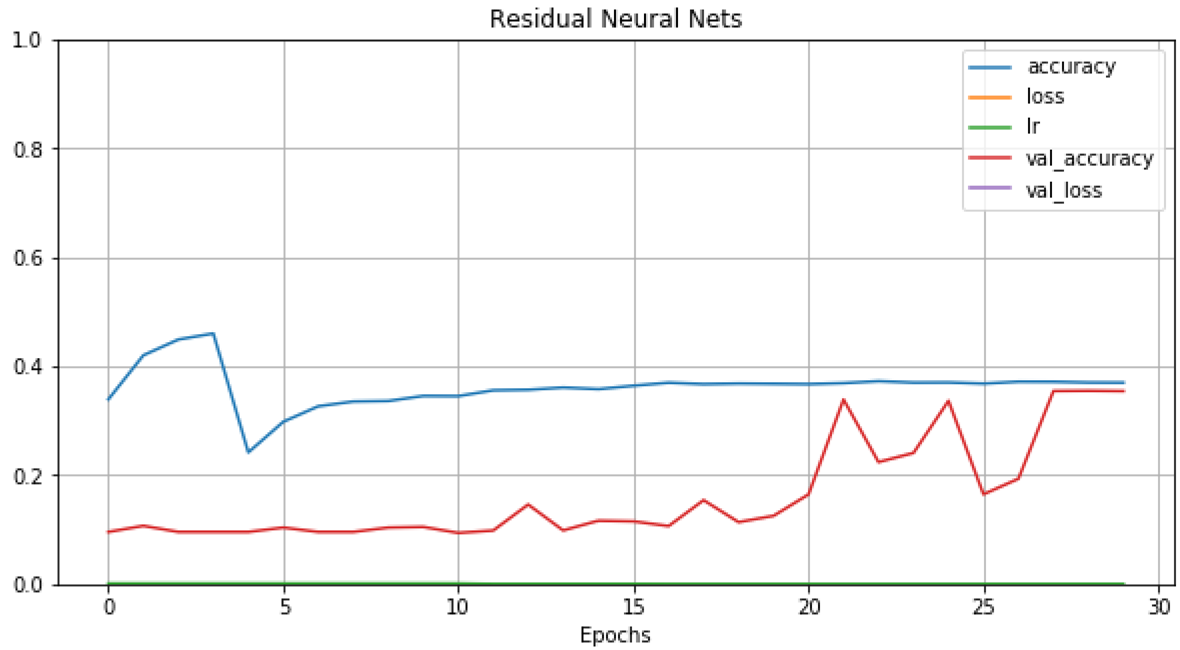
```
Learning rate:  1e-06
Epoch 22/30
45000/45000 [==============================] - 333s 7ms/sample - loss: 1.8171 - accuracy: 0.3693 - val_loss:
1.9610 - val_accuracy: 0.3392
Learning rate:  1e-06
Epoch 23/30
45000/45000 [==============================] - 339s 8ms/sample - loss: 1.8119 - accuracy: 0.3729 - val_loss:
4.0643 - val_accuracy: 0.2242
Learning rate:  1e-06
Epoch 24/30
45000/45000 [==============================] - 341s 8ms/sample - loss: 1.8122 - accuracy: 0.3703 - val_loss:
3.2406 - val_accuracy: 0.2410
Learning rate:  1e-06
Epoch 25/30
45000/45000 [==============================] - 341s 8ms/sample - loss: 1.8136 - accuracy: 0.3705 - val_loss:
1.9577 - val_accuracy: 0.3368
Learning rate:  1e-06
Epoch 26/30
45000/45000 [==============================] - 344s 8ms/sample - loss: 1.8200 - accuracy: 0.3684 - val_loss:
10.5099 - val_accuracy: 0.1650
Learning rate:  5e-07
Epoch 27/30
45000/45000 [==============================] - 323s 7ms/sample - loss: 1.8193 - accuracy: 0.3720 - val_loss:
5.0268 - val_accuracy: 0.1936
Learning rate:  5e-07
Epoch 28/30
45000/45000 [==============================] - 337s 7ms/sample - loss: 1.8104 - accuracy: 0.3718 - val_loss:
1.8643 - val_accuracy: 0.3548
Learning rate:  5e-07
Epoch 29/30
45000/45000 [==============================] - 339s 8ms/sample - loss: 1.8162 - accuracy: 0.3705 - val_loss:
1.8718 - val_accuracy: 0.3552
Learning rate:  5e-07
Epoch 30/30
45000/45000 [==============================] - 358s 8ms/sample - loss: 1.8195 - accuracy: 0.3702 - val_loss:
1.8797 - val_accuracy: 0.3546
```

Out[50]: `<matplotlib.axes._subplots.AxesSubplot at 0x26de6928e48>`

Out[50]: `(0, 1)`

Out[50]: `Text(0.5, 0, 'Epochs')`

Out[50]: `Text(0.5, 1.0, 'Residual Neural Nets')`




```
10000/10000 [==============================] - 13s 1ms/sample - loss: 1.8755 - accuracy: 0.3555
```

Out[50]: [1.875451368713379, 0.3555]

```
Model: "resnet"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
img (InputLayer)             [(None, 32, 32, 3)]       0         
_________________________________________________________________
dense_124 (Dense)            (None, 32, 32, 128)       512       
_________________________________________________________________
dense_125 (Dense)            (None, 32, 32, 128)       16512     
_________________________________________________________________
dense_126 (Dense)            (None, 32, 32, 128)       16512     
_________________________________________________________________
batch_normalization_56 (Batc (None, 32, 32, 128)       512       
_________________________________________________________________
activation_56 (Activation)   (None, 32, 32, 128)       0         
_________________________________________________________________
dropout_59 (Dropout)         (None, 32, 32, 128)       0         
_________________________________________________________________
dense_127 (Dense)            (None, 32, 32, 128)       16512     
_________________________________________________________________
batch_normalization_57 (Batc (None, 32, 32, 128)       512       
_________________________________________________________________
activation_57 (Activation)   (None, 32, 32, 128)       0         
_________________________________________________________________
dropout_60 (Dropout)         (None, 32, 32, 128)       0         
_________________________________________________________________
flatten_15 (Flatten)         (None, 131072)            0         
_________________________________________________________________
dense_128 (Dense)            (None, 10)                1310730   
=================================================================
Total params: 1,361,802
Trainable params: 1,361,290
Non-trainable params: 512
_________________________________________________________________
```

Analysis:

- We can see the performance of Residual nets are good compared to normal functional API. Accuracies are imoproved using Residual nets.
- By Increasing number trainable layers, training time is increasing significantly per epoch. But it might increase the accuracy slightly.
- I think, we need to use convolute neural net filters with proper hyper parameters to improve the accuracy.
- I tried, to tune the learning rate by using call back, but without proper filter, there is not increase in the accuracy.
- I think, we need to use convoluted neural nets with proper learning rate (Max learning rate/2). Max learning rate is the rate at which, acurracy declines.
- I tried batch sizes 128 and 32 but it did not impact the accuracy. I think, batch size would be subjective to other hyperparameters like learning rate and epochs.
- I tried, epochs 20 and 30. Both gave same performance. High value for epochs is not attempted because of high train speed. We can use early stopping call back to reduce training speed.

Step 11: Residual nets and gradient descent

- In a neural network, Gradient descent is used to minizie the error function and it is used during Forward propagation and Backworkd propagation.
- Forward propagation: Gradient descent is used to compute and pass the parameters to next layer. Parameters are calculated by minimize the cost function. Gradient descent is the method for minimization. In residual nets, residual layer also learns from the gradient and it will be added at the connection.
- Backward propagation: Gradient descent is used to update the parameters to train the network. In residual nets, residual layer are updated by the gradients and contribute in updaing the parameters along the layer.
- Residual nets are used to pass the actual domain knowledge of input to next layers without manipuation, by skipping. In order to make the model better, we need to tune the hyperparameters of the module and more deepers layers should be added.

Step 12: Res nets in project

- Pros of using Res nets

  1. Our data has a certain level of complexity. (Not linearly seprable). Training data is not huge. Using Res Nets, we can train better.
- Cons of using Res net

  1. As of data do not deal with images, number of input features is considerbaly less. As Residual nets have higher training speeds, compared to sequential and functional nets, we need to validate the performance of Functional API before using Residual nets. There is a chance that complexith of our data might not demand residual nets which would save the training speeds.

In [ ]:

```

```